



**EMBARCATEH
CAMPINAS - SP**

Projeto:
***Sistema para Monitoramento Inteligente de Movimentos
em Contêineres Marítimos com TinyML***

Adriana Rocha Castro de Paula
Elias Kento Tomiyama
Vagner S. Vasconcelos
Setembro de 2025

1. Contexto

A logística de contêineres marítimos é um componente crítico da cadeia de suprimentos global, envolvendo etapas como movimentação por empilhadeiras ou guindastes, transporte terrestre em caminhões e transporte marítimo em navios. A rastreabilidade precisa dessas etapas é essencial para otimizar rotas, reduzir custos operacionais, prevenir danos causados por vibrações excessivas e detectar anomalias em tempo real. No entanto, sistemas tradicionais baseados em GPS e registros manuais carecem de granularidade para identificar o tipo de movimento do contêiner, limitando a capacidade de resposta proativa.

A **concepção** deste projeto é criar um sistema embarcado de baixo custo e alta autonomia, embarcado no próprio contêiner, que utilize sensores inerciais e inteligência artificial na borda (*edge*) para resolver esse problema. A ideia é "dar inteligência" ao contêiner, permitindo que ele "entenda" seu próprio estado de movimento e comunique essa informação em tempo real. A solução se baseia no campo emergente de *Tiny Machine Learning* (TinyML), que foca na implementação de modelos de aprendizado de máquina em microcontroladores com recursos limitados.

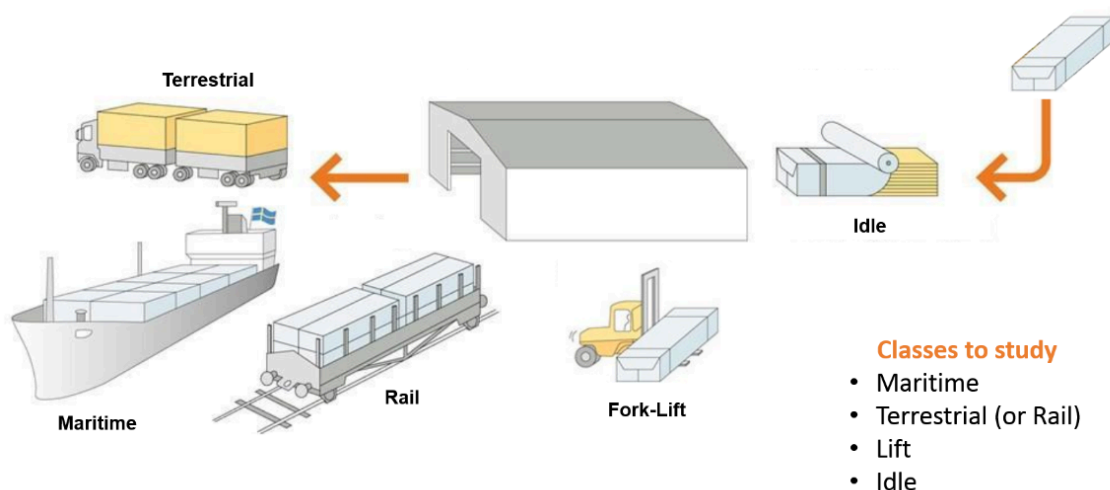
TinyML permite a execução de modelos de aprendizado de máquina em microcontroladores com recursos limitados, como o RP2040 da plataforma BitDogLab. Este projeto propõe um sistema embarcado que utiliza o acelerômetro MPU6500, a plataforma Edge Impulse para treinamento de modelos, e o protocolo MQTT para transmissão de dados, classificando automaticamente os movimentos de contêineres em quatro classes:

(i) parado (armazenado ou em espera), (ii) subindo/descendo (movimentação por empilhadeira ou guindaste), (iii) esquerda/direita (transporte terrestre), e (iv) ziguezague (transporte marítimo).

O sistema opera com baixo consumo energético, exibe resultados localmente em um display SSD1306 e transmite dados via Wi-Fi, alinhando-se aos objetivos do curso Embarcatech de criar soluções embarcadas inovadoras para desafios reais. A figura 1, ilustra o estudo de caso em questão:

Figura 1 - Caso de Uso. Fonte: Curso UNIFEI-IESTI01- TinyML

Case Study: Mechanical Stresses in Transport



2. Usuários

O sistema proposto oferece utilidade direta para múltiplos atores da cadeia logística, gerando valor econômico e operacional.

Os usuários-alvo do sistema incluem:

- **Operadores logísticos:** Gerenciam cadeias de suprimentos e se beneficiam de dados em tempo real para otimizar rotas e reduzir custos (a automação do registro de etapas (ex: início do transporte terrestre) elimina a necessidade de apontamentos manuais, reduzindo erros e custos administrativos).
- **Gestores de armazéns e portos:** Necessitam de monitoramento preciso para prevenir danos durante o manuseio e melhorar a segurança (a detecção de padrões de vibração anômalos pode indicar manuseio incorreto ou tentativa de violação, disparando alertas imediatos).
- **Desenvolvedores de sistemas embarcados:** Podem usar o projeto como base para soluções escaláveis, integrando novos sensores ou funcionalidades.
- **Empresas de transporte marítimo:** Interessadas em rastreabilidade detalhada para detectar anomalias e melhorar a eficiência operacional.
- **Para o Ecossistema de Cidades Inteligentes:**
 - A coleta de dados em massa sobre o fluxo de contêineres pode alimentar sistemas de gerenciamento de tráfego e planejamento urbano, otimizando as rotas de veículos pesados.

3. Objetivo Geral

Desenvolver um sistema embarcado baseado na plataforma BitDogLab, utilizando TinyML e o acelerômetro MPU6500, para classificar automaticamente os movimentos de contêineres marítimos em quatro classes (parado, subindo/descendo, esquerda/direita, ziguezague), com visualização local em um display SSD1306, transmissão de dados via MQTT para monitoramento remoto, e operação otimizada para baixo consumo energético, alcançando autonomia mínima de 30 dias.

4. Objetivos Específicos

1. Coletar dados do acelerômetro MPU6500 (eixos X, Y, Z) a 60 Hz para capturar padrões de movimento.
2. Pré-processar os dados com filtro passa-baixa, normalização e janelamento para preparar os sinais para o modelo TinyML.
3. Desenvolver, treinar e avaliar um modelo de aprendizado de máquina na plataforma Edge Impulse, alcançando acurácia superior a 80%.
4. Implementar o modelo TinyML na BitDogLab para inferência em tempo real com latência inferior a 200 ms.
5. Exibir a classe de movimento detectada em um display OLED SSD1306.
6. Transmitir os resultados via Wi-Fi para um broker MQTT com qualidade de serviço (QoS) 2.

7. Garantir operação em modo de baixo consumo para autonomia mínima de 30 dias com bateria.
8. Utilizar FreeRTOS para gerenciar tarefas concorrentes, assegurando estabilidade e escalabilidade.

5. Requisitos

5.1 Requisitos Funcionais

- **RF01:** Coletar dados do acelerômetro MPU6500 (eixos X, Y, Z) a 60 Hz.
- **RF02:** Pré-processar os dados com filtro passa-baixa, normalização e janelamento.
- **RF03:** Executar um modelo TinyML para classificação em tempo real das quatro classes de movimento.
- **RF04:** Exibir a classe detectada no display OLED SSD1306.
- **RF05:** Transmitir resultados via Wi-Fi para um broker MQTT em nuvem, com QoS 2 e estampa de tempo (dia/mês/ano - hora:minuto: segundo).
- **RF06:** Operar em modo de baixo consumo quando o contêiner estiver parado.
- **RF07:** Utilizar FreeRTOS para gerenciar tarefas concorrentes.

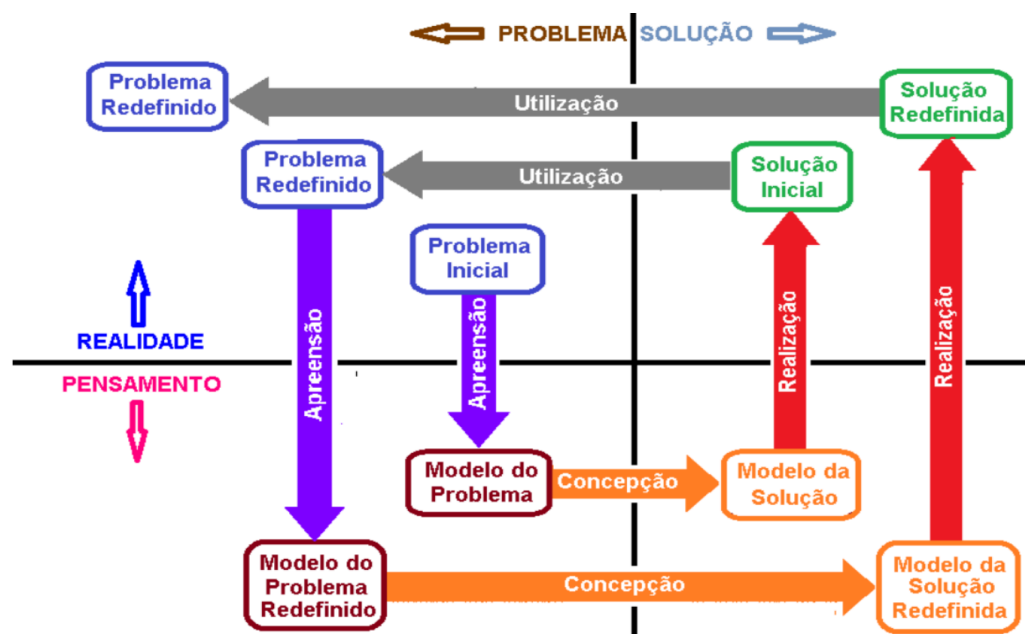
5.2 Requisitos Não Funcionais

- **RNF01:** Acurácia do modelo superior a 80% em condições reais (dados de teste).
- **RNF02:** Latência de inferência inferior a 200 ms.
- **RNF03:** Autonomia mínima de 30 dias com bateria.
- **RNF04:** Funcionamento estável em temperaturas de 0°C a 50°C.
- **RNF05:** Código modular, documentado e compatível com boas práticas de manutenção.

6. Abordagem

O projeto segue a metodologia CUGNASA, estruturada em quatro etapas, com atividades divididas em duas frentes: desenvolvimento do sistema embarcado e desenvolvimento do modelo TinyML. A abordagem é iterativa, com ciclos de projeto, prototipagem, teste e refinamento, utilizando ferramentas como Pico-SDK, FreeRTOS e Edge Impulse.

Figura 2 -
Metodologia.
Fonte: Cugnasa
(2025)



6.1 Etapa 1: Definição de Requisitos e Lista de Materiais

Objetivo: Consolidar o problema, definir requisitos e listar materiais necessários.

Atividades:

- **Documentando o problema:** O rastreamento de contêineres carece de granularidade para identificar tipos de movimento, impactando eficiência e segurança. A solução baseada em TinyML e IoT permite monitoramento detalhado e em tempo real.
- **Detalhando requisitos:** Definir RFs e RNFs com base no contexto logístico, garantindo alinhamento com as necessidades dos usuários.
- **Listando materiais:**
 - **Hardware:**
 - BitDogLab (RP2040): Microcontrolador principal.
 - MPU6500: Acelerômetro de 6 eixos (I2C).
 - SSD1306: Display OLED 128x64 (I2C).
 - Módulo Wi-Fi: ESP8266 ou integrado à BitDogLab (SPI).
 - Bateria: LiPo 3.7V, capacidade ~2000 mAh (a ser confirmada).
 - **Software:**
 - Pico-SDK: Framework para programação do RP2040.
 - FreeRTOS: Gerenciamento de tarefas concorrentes.
 - Edge Impulse: Desenvolvimento do modelo TinyML.
 - Paho MQTT: Biblioteca para comunicação MQTT.
- **Integração:** conexão dos periféricos (I2C para MPU6500 e SSD1306, SPI para Wi-Fi) e compatibilidade com o RP2040.

6.2 Etapa 2: Arquitetura e Modelagem

Objetivo: Definir a arquitetura do sistema e modelar o software e o modelo TinyML.

Atividades:

- **Diagrama de hardware:**
 - **Conexões:**
 1. MPU6500: I2C (SDA: GPIO4, SCL: GPIO5).
 2. SSD1306: I2C (mesmo barramento, endereços distintos).
 3. Módulo Wi-Fi: SPI (MOSI: GPIO7, MISO: GPIO8, SCK: GPIO6, CS: GPIO9).
 4. Bateria: Conectada ao regulador de tensão da BitDogLab.

Ambos os componentes utilizados (Acelerômetro MPU6500 e Display SSD1306) possuem interface serial de comunicação do tipo I2C. Na BitDogLab, o display já está conectado na interface I2C1 (GPIO 14 e 15); conectaremos o acelerômetro na interface I2C0 (GP0 e GP1), sendo o RP2040, em ambos o caso, o mestre da rede e os componentes os escravos.

O módulo wi-fi (CYW43439) já é integrado a placa Pi Pico W, disponibilizada na BitDogLab.

Endereços de rede:

Componente	Endereço
MPU6050	0x68
SSD1306	0x3C

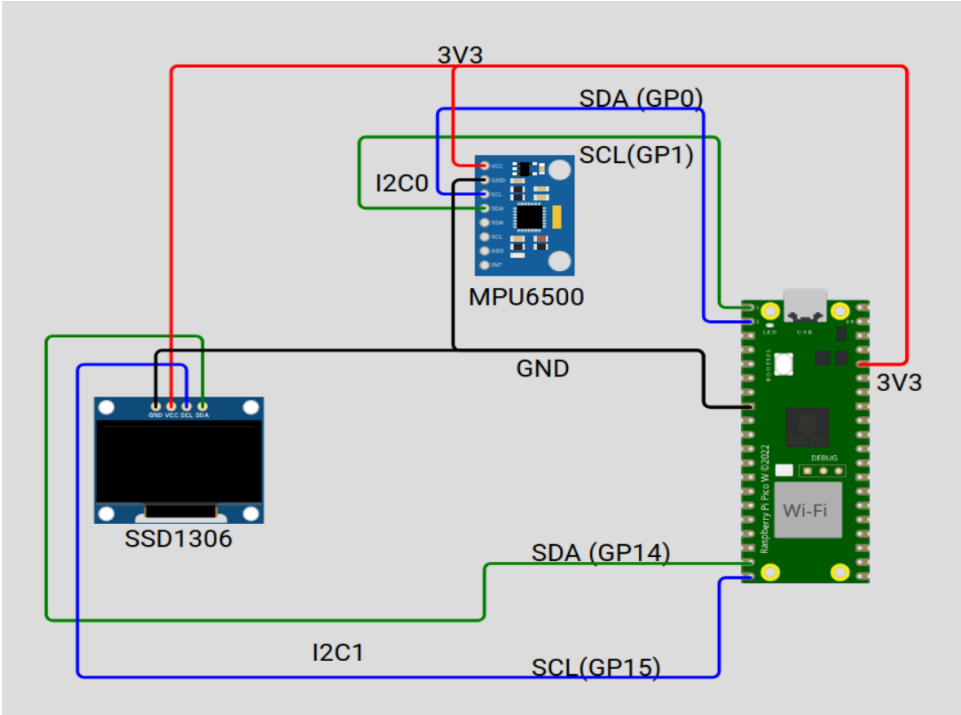


figura 3 - diagrama de hardware proposto

Blocos funcionais do software

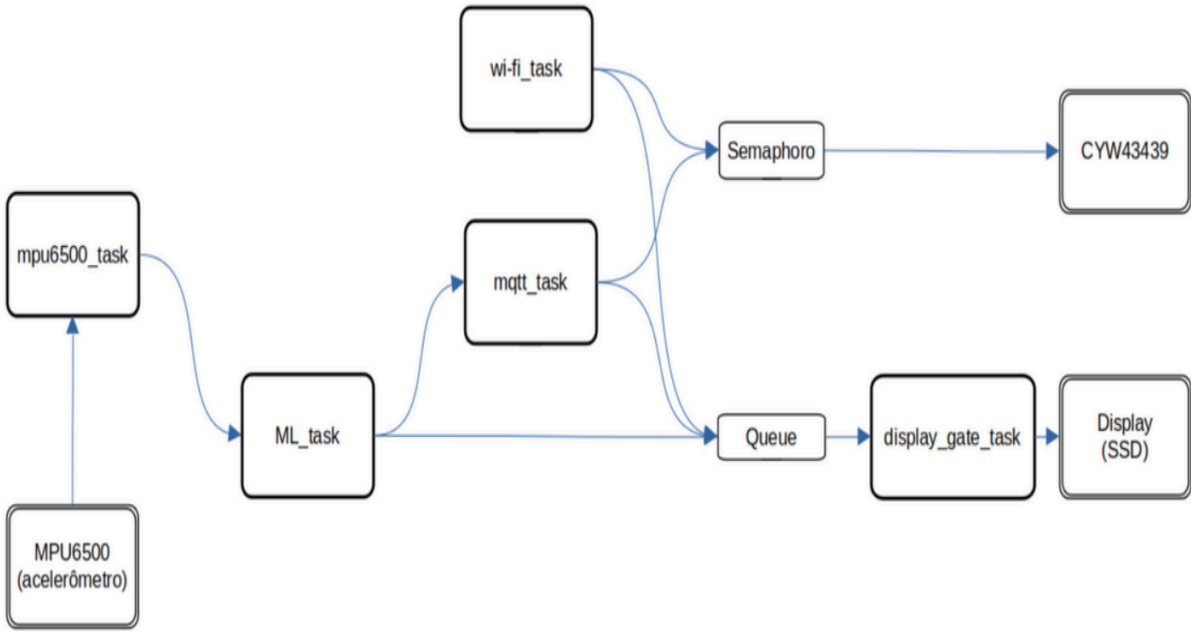


figura 4 - diagrama de blocos funcionais do software proposto.

Conforme apresentado na figura 4, a proposta envolve a utilização de um sistema operacional de tempo real (FreeRTOS) para o controle do sistema. Serão desenvolvidas as seguintes tarefas:

Tarefa	Função
Wi-fi_task	Realiza autenticação no ponto de acesso wi-fi
mqtt_task	Envia os dados para o broker mqtt na nuvem e para a tarefa display_gate_task
display_gate_task	Envia os dados para o display
mpu6500_task	Lê os dados do acelerômetro e os envia para a tarefa ML_task
ML_task	Recebe os dados do acelerômetro e realiza a classificação do movimento (inferência) e envia o resultado para o wi-fi e display

Temos ainda o Semaphoro, que atua como mecanismo de sinalização entre as tarefas: Wi-fi_task e mqtt_task, de forma que a tarefa mqtt_task aguarda a autenticação da Wi-fi_task para iniciar; e a fila Queue que controla o acesso ao display.

Explicação da Lógica e Estrutura do Sistema

A arquitetura de software proposta foi projetada para ser modular, escalável e robusta, atendendo a todos os requisitos funcionais e não funcionais do projeto. A escolha central da arquitetura é o uso de um Sistema Operacional de Tempo Real (RTOS), especificamente o FreeRTOS, que permite gerenciar as múltiplas atividades do sistema de forma concorrente e organizada.

Arquitetura Baseada em Tarefas Concorrentes

O sistema é decomposto em cinco tarefas principais, cada uma com uma responsabilidade única. Essa abordagem, fundamentada no FreeRTOS, permite que operações lentas (como comunicação Wi-Fi) não bloqueiem operações críticas de tempo real (como a aquisição de dados do sensor e a inferência do modelo).

mpu6500_task: Sua única função é coletar dados do acelerômetro MPU6050 na frequência de 60 Hz estipulada no requisito RF01. Ela opera com alta prioridade para garantir a precisão temporal da amostragem.

ML_task: É o núcleo de inteligência do sistema. Recebe os dados brutos da mpu6050_task, realiza o pré-processamento (RF02) e executa a inferência do modelo TinyML (RF03) para classificar o

movimento. Esta tarefa é projetada para ser computacionalmente intensiva, mas sua execução em paralelo garante que a latência de inferência seja mantida abaixo de 200 ms (RNF02).

wifi_task: Gerenciar a conectividade Wi-Fi. Sua responsabilidade é estabelecer e manter a conexão com o ponto de acesso.

mqtt_task: Responsável por transmitir os resultados da classificação para um broker MQTT na nuvem, atendendo ao requisito RF05. Ela formata a mensagem com a classe detectada e uma estampa de tempo, utilizando QoS 2 para garantir a entrega confiável da mensagem.

display_gate_task: Controla a exibição das informações no display OLED SSD1306 (RF04). Funciona como um "gatekeeper" para o periférico, evitando conflitos de acesso e desacoplando a lógica de exibição do processamento principal.

Mecanismos de Sincronização e Comunicação Inter-Tarefas

Para garantir a operação correta e a integridade dos dados, a comunicação entre as tarefas é gerenciada por mecanismos específicos do FreeRTOS:

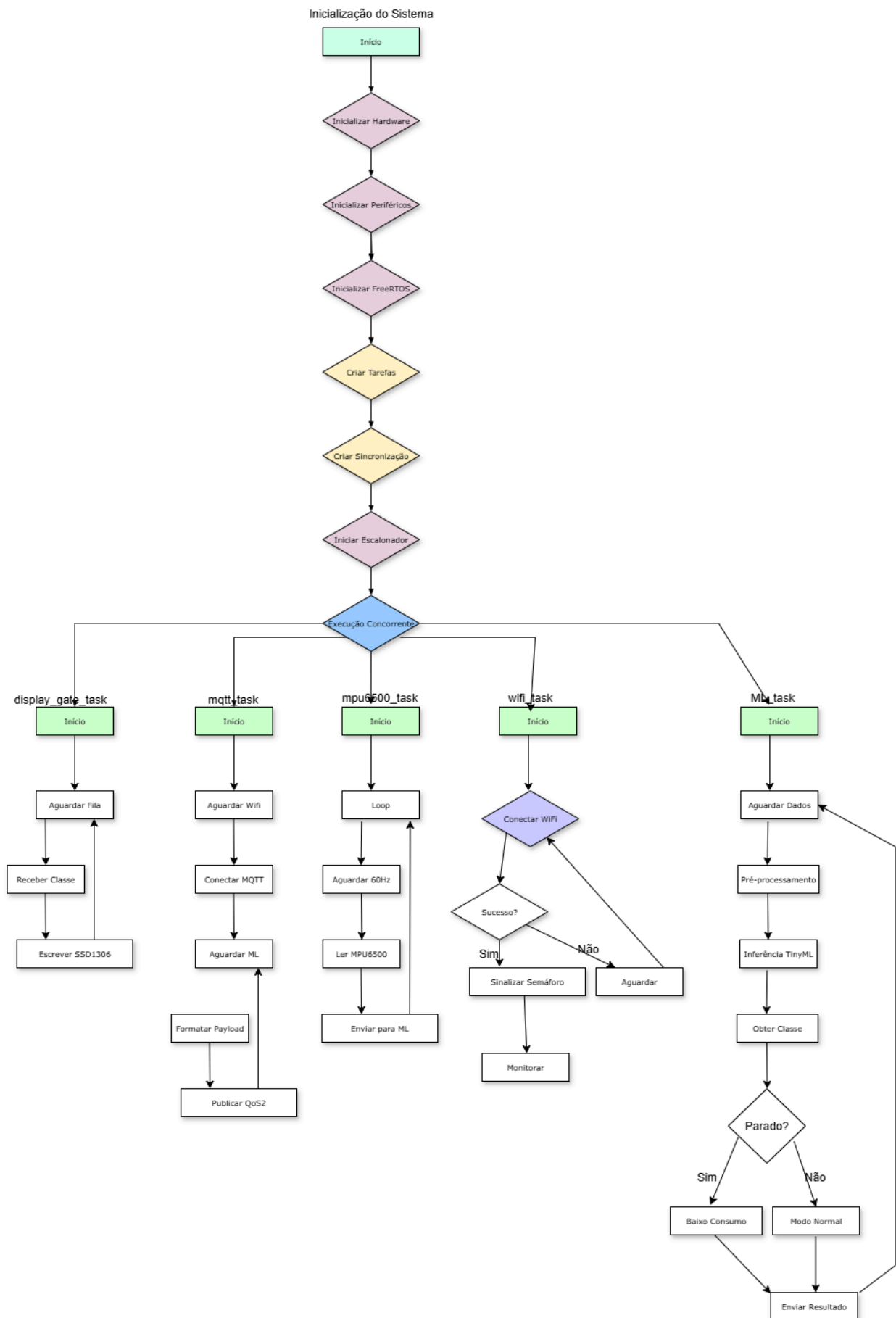
Semáforo (entre wifi_task e mqtt_task): Conforme definido no diagrama de blocos, um semáforo atua como um mecanismo de sinalização. A mqtt_task deve aguardar um sinal (a "liberação" do semáforo) da wifi_task antes de tentar se conectar ao broker e enviar dados. Isso garante que a transmissão MQTT só seja iniciada após a confirmação de uma conexão Wi-Fi ativa, prevenindo falhas e otimizando recursos.

Fila (para a display_gate_task): A ML_task, após cada inferência, envia o resultado (a classe do movimento) para uma fila (Queue). A display_gate_task fica bloqueada aguardando a chegada de um item nessa fila. Essa estrutura desacopla o processamento da exibição: a ML_task não precisa esperar a lenta atualização do display I2C para continuar seu trabalho. A fila também serializa o acesso ao display, garantindo que as informações sejam exibidas de forma ordenada e sem corrupção.

Lógica de Baixo Consumo (Atendendo RNF03)

A lógica para atingir a autonomia de 30 dias é um pilar central do sistema. Ela é implementada na ML_task e atende ao requisito RF06. Quando a inferência resulta na classe "Parado", a tarefa inicia um protocolo de economia de energia. Este protocolo pode incluir a redução da frequência de amostragem da mpu6500_task (já que não são esperadas mudanças bruscas) e a desativação temporária do módulo Wi-Fi, que é um dos maiores consumidores de energia. O sistema permanecerá neste estado de baixo consumo até que o acelerômetro detecte uma vibração que justifique reativar o modo de operação normal, garantindo que o dispositivo economize bateria significativamente durante os longos períodos de espera do contêiner. Em suma, a estrutura proposta utiliza os pontos fortes do FreeRTOS para criar um sistema embarcado que é ao mesmo tempo reativo, eficiente e robusto, alinhado com todos os objetivos e requisitos definidos para o projeto.

Fluxograma do software



6.3 Etapa 3: Prototipagem e Ajustes

Objetivo: Construir e testar o protótipo funcional, identificando ajustes necessários.

Atividades:

- **Sistema Embarcado:**
 - **Configurando FreeRTOS:** Implementar tarefas concorrentes:
 - Tarefa 1: Aquisição de dados (MPU6500, 60 Hz, buffer circular).
 - Tarefa 2: Pré-processamento (filtro passa-baixa, normalização, janelamento).
 - Tarefa 3: Inferência TinyML.
 - Tarefa 4: Exibição no SSD1306.
 - Tarefa 5: Transmissão MQTT (QoS 2).
 - Tarefa 6: Gerenciamento de energia (modo sleep).
 - **Implementando aquisição de dados:** Configurar MPU6500 via I2C, ajustando escala ($\pm 2g$) e frequência (60 Hz).
 - **Configurando display:** Inicializar SSD1306 para exibir a classe detectada (ex.: "Parado", "Subindo", etc.).
 - **Implementando MQTT:** Configurar cliente Paho MQTT, conectar ao broker (ex.: Mosquitto) e publicar mensagens com QoS 2.
 - **Otimizando energia:** Ativar modo sleep do RP2040 quando parado, reduzindo frequência de amostragem e desativando Wi-Fi.
- **Modelo TinyML:**
 - **Coletando dados:** Simular movimentos em laboratório (parado, subindo/descendo com elevação manual, esquerda/direita com deslocamento linear, ziguezague com oscilações). Coletar 100 amostras por classe (1 min cada, 60 Hz).
 - **Pré-processando no Edge Impulse:**
 - Filtro passa-baixa: Média móvel, frequência de corte 10 Hz.
 - Normalização: Escala [0,1] com base no intervalo de aceleração ($\pm 2g$).
 - Janelamento: Janelas de 1s (60 amostras), sobreposição de 50%.
 - **Projetando modelo:** Rede neural densa (32 neurônios x 2 camadas, ReLU, softmax).
 - **Treinando e avaliando:** Usar validação cruzada, ajustando hiperparâmetros para acurácia > 80%.
 - **Convertendo modelo:** Exportar para C++ (TensorFlow Lite Micro) e integrar ao firmware.
 - **Executando inferência:** Testar latência no RP2040 (< 200 ms).
- **Testes:**
 - **Unitários:** Verificar cada tarefa (aquisição, pré-processamento, inferência, exibição, transmissão).
 - **Integração:** Testar interação entre tarefas e comunicação MQTT.

- **Campo:** Simular movimentos reais (ex.: deslocar BitDogLab em carrinho para esquerda/direita, elevar para subindo/descendo).
- **Métricas:**
 - Acurácia: Comparar previsões com rótulos reais.
 - Latência: Medir tempo de inferência.
 - Consumo: Monitorar corrente com multímetro.
- **Documentando desafios:** Ex.: ruído no MPU6500, falhas Wi-Fi, ajustes no modelo.

6.4 Etapa 4: Entrega Final e Documentação

Objetivo: Finalizar o projeto com ajustes implementados e documentação completa.

Atividades:

- **Implementando ajustes:**
 - Otimizar modelo TinyML (ex.: reduzir camadas ou neurônios para menor latência).
 - Ajustar consumo energético (ex.: amostragem adaptativa baseada no estado).
 - Corrigir falhas identificadas (ex.: reconexão automática ao broker MQTT).
- **Otimizando energia:**
 - Reduzir frequência de amostragem para 10 Hz quando parado.
 - Desativar Wi-Fi e display em modo sleep.
 - Estimar autonomia com bateria de 2000 mAh (ex.: 50 μ A em sleep, 100 mA em operação).
- **Finalizando documentação:**
 - Código comentado, organizado em módulos (aquisição, inferência, comunicação).
 - Diagramas atualizados (hardware, software).
 - Relatório técnico com resultados de testes (acurácia, latência, consumo).
- **Publicando no GitHub:**

Criar repositório com estrutura:

Estrutura

```

|— CMakeLists.txt
|— pico_sdk_import.cmake
|— src/           # código-fonte (.c/.cpp)
|— include/       # headers (.h/.hpp)
|— lib/           # bibliotecas de terceiros e código compartilhado
|— docs/          # documentos auxiliares
|— scripts/       # scripts (py/sh/ps1/bat)
|— tests/         # testes
|— .vscode/       # config para IntelliSense
|— .gitignore

```

LICENSE

README com instruções de compilação, execução e uso.

Entregável: Sistema funcional, documentação completa (relatório, diagramas, código), repositório GitHub.

7. Solução

7.1 Hardware

- **BitDogLab (RP2040):** Microcontrolador principal, gerencia tarefas e comunicação.
- **MPU6500:** Acelerômetro de 6 eixos (I2C), coleta dados a 60 Hz.
- **SSD1306:** Display OLED 128x64 (I2C), exibe classe de movimento.
- **Módulo Wi-Fi:** ESP8266 (SPI), transmite dados via MQTT.
- **Bateria:** LiPo 3.7V, 2000 mAh, para autonomia de 30 dias.

7.2 Software

- **Pico-SDK:** Configuração de periféricos e comunicação.
- **FreeRTOS:** Gerenciamento de tarefas concorrentes:
 - **Tarefa 1:** Aquisição de dados (MPU6500, buffer circular).
 - **Tarefa 2:** Pré-processamento (filtro passa-baixa, normalização, janelamento).
 - **Tarefa 3:** Inferência TinyML.
 - **Tarefa 4:** Exibição no SSD1306.
 - **Tarefa 5:** Transmissão MQTT (Paho, QoS 2).
 - **Tarefa 6:** Gerenciamento de energia (modo sleep).
- **Edge Impulse:** Desenvolvimento do modelo TinyML.

Don't miss the unveiling of HiveMQ Pulse with Walker Reynolds! [Join the webinar](#) →

HIVEMQ Products Solutions MQTT Learn Company Pricing Login [Start Free](#) [Contact Us](#)


Unlock the value of your Logistics

Securely collect, stream, and govern industrial data in real-time for smarter decision-making

[Start Free](#) [Contact Us](#)

Integrate OT/IT
Data Driven Backbone
Unlock Data Value
AI-Ready

Ask AI


HiveMQ Cloud
Q Search i

ORGANIZATIONS

Adriana's organ ▾

👤

CLUSTERS OVERVIEW +

Free #1

Documentation ↗

Free #1

Overview
Access Management
Integrations
Web Client
Getting Started

Please connect to the WebClient in order to subscribe to topics

Connection Settings

Connect to your HiveMQ Cloud Cluster with your credentials. Do not worry you can quickly connect with autogenerated credentials.

Username *
Password *

IoTmyDevices

.....

Connect

or

Connect with autogenerated credentials

Topic Subscriptions

Unsubscribe from all topics

Subscribe to topics to receive messages from the HiveMQ cluster. You can also set the Quality of Service (QoS) for each topic. The higher the QoS, the more reliable the message delivery is. You can always

Messages

Pleas
incon

```
#include "MQTTClient.h"
#include "pico/cyw43_arch.h"
#include "pico/stdlib.h"
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "queue.h"
#include <stdio.h>
```

```
// Inclusão dos cabeçalhos da biblioteca Paho MQTT e das nossas implementações de rede.
#include "paho_network.h"
#include "paho_timer.h"
```

```
// =====
// CONFIGURAÇÕES DE REDE e MQTT
// =====
```

```
#define WIFI_SSID      "ALEX_2G"
#define WIFI_PASSWORD  "12345678"
```

```
// Servidor MQTT (SUBSTITUIR PELO SEU BROKER)
```

```

// #define MQTT_BROKER_HOST      "seu_broker.s1.eu.hivemq.cloud"
#define MQTT_BROKER_HOST      "test.mosquitto.org"
#define MQTT_BROKER_PORT      1883
// #define MQTT_BROKER_PORT_SSL  8883    // Porta SSL/TLS

// Configurações de autenticação MQTT
#define MQTT_USE_AUTH          0    // true para habilitar autenticação
#define MQTT_USERNAME          "lotmyDevices"
#define MQTT_PASSWORD          "senha"

// Configurações de conexão MQTT
#define MQTT_CLIENT_ID_PREFIX  "tinymI_gate_"
#define MQTT_CLIENT_ID         "tinymI_gate_pico_001"
#define MQTT_KEEP_ALIVE_SEC    60    // Keep alive: 60 segundos
#define MQTT_CONNECT_TIMEOUT_MS 10000 // Timeout conexão: 10s
#define MQTT_QOS_LEVEL         0    // QoS 0: At most once delivery

// Tópicos MQTT
#define MQTT_TOPIC_ROOT        "tinymI/iot/myDevices"
#define MQTT_TOPIC_DATA        MQTT_TOPIC_ROOT "/data"
#define MQTT_TOPIC_ALERTS      MQTT_TOPIC_ROOT "/alerts"
#define MQTT_TOPIC_STATUS      MQTT_TOPIC_ROOT "/status"
#define MQTT_TOPIC_CONFIG      MQTT_TOPIC_ROOT "/config"
#define MQTT_TOPIC_COMMANDS    MQTT_TOPIC_ROOT "/commands"

// Para teste
#define MQTT_SUB_TOPIC          MQTT_TOPIC_COMMANDS
#define MQTT_PUB_TOPIC          MQTT_TOPIC_STATUS

// Configurações de buffer MQTT
#define MQTT_BUFFER_SIZE        1024    // Buffer para mensagens MQTT
#define MQTT_MAX_MESSAGE_SIZE   512    // Tamanho máximo de mensagem

// Buffer de rede e MQTT
static unsigned char sendbuf[512];
static unsigned char readbuf[512];

// Função para verificar status do WiFi
static bool check_wifi_status(void) {
    int status = cyw43_wifi_link_status(&cyw43_state, CYW43_ITF_STA);
    printf("WiFi status: %d\n", status);
    return (status == CYW43_LINK_UP);
}

```

```

// Função callback para mensagens recebidas
void messageArrived(MessageData* data) {
    printf("=== Mensagem MQTT Recebida ===\n");
    printf("Tópico: %.*s\n", data->topicName->lenstring.len, data->topicName->lenstring.data);
    printf("Payload: %.*s\n", (int)data->message->payloadlen, (char*)data->message->payload);
    printf("=====\n");
}

// Task MQTT
void mqtt_task(void *pvParameters) {
    int rc = 0;
    Network network;
    MQTTClient client;
    MQTTPacket_connectData connectData = MQTTPacket_connectData_initializer;

    printf("\n=== MQTT Task Iniciada ===\n");
    printf("Stack livre: %d bytes\n", uxTaskGetStackHighWaterMark(NULL) * sizeof(StackType_t));
    printf("Heap livre: %d bytes\n", xPortGetFreeHeapSize());

    // Aguarda WiFi estar conectado
    printf("Aguardando WiFi estar conectado...\n");
    int wifi_retry = 0;
    while (!check_wifi_status() && wifi_retry < 30) {
        printf("WiFi não conectado, tentativa %d/30\n", wifi_retry + 1);
        vTaskDelay(pdMS_TO_TICKS(1000));
        wifi_retry++;
    }

    if (wifi_retry >= 30) {
        printf("ERRO: WiFi não conectou após 30 tentativas\n");
        vTaskDelete(NULL);
        return;
    }

    printf("WiFi conectado! Prosseguindo com MQTT...\n");

    // Inicializa rede usando nossa implementação
    printf("Inicializando estrutura de rede...\n");
    NetworkInit(&network);

    printf("Tentando conectar ao broker %s:%d\n", MQTT_BROKER_HOST, MQTT_BROKER_PORT);

```

```

if ((rc = NetworkConnect(&network, MQTT_BROKER_HOST, MQTT_BROKER_PORT)) != 0) {
    printf("ERRO: Falha ao conectar no broker %s:%d (código: %d)\n", MQTT_BROKER_HOST,
MQTT_BROKER_PORT, rc);
    vTaskDelete(NULL);
    return;
}
printf("✓ Conectado ao broker TCP %s:%d\n", MQTT_BROKER_HOST, MQTT_BROKER_PORT);

// Inicializa cliente MQTT
printf("Inicializando cliente MQTT...\n");
MQTTClientInit(&client, &network, MQTT_CONNECT_TIMEOUT_MS, sendbuf, sizeof(sendbuf),
readbuf, sizeof(readbuf));

// Configura dados de conexão
connectData.MQTTVersion = 3;
connectData.clientID.cstring = MQTT_CLIENT_ID;
connectData.keepAliveInterval = MQTT_KEEP_ALIVE_SEC;
connectData.cleansession = 1;

// Configurar autenticação se habilitada
#if MQTT_USE_AUTH
connectData.username.cstring = MQTT_USERNAME;
connectData.password.cstring = MQTT_PASSWORD;
printf("Usando autenticação: usuário=%s\n", MQTT_USERNAME);
#endif

printf("Conectando cliente MQTT (timeout: %dms)...\n", MQTT_CONNECT_TIMEOUT_MS);
if ((rc = MQTTConnect(&client, &connectData)) != 0) {
    printf("ERRO: Falha ao conectar MQTT, código %d\n", rc);
    NetworkDisconnect(&network);
    vTaskDelete(NULL);
    return;
}
printf("✓ Cliente MQTT conectado com sucesso!\n");

// Subscribe em um tópico
printf("Subscribendo em %s...\n", MQTT_SUB_TOPIC);
if ((rc = MQTTSubscribe(&client, MQTT_SUB_TOPIC, QOS0, messageArrived)) != 0) {
    printf("ERRO: Falha ao inscrever, código %d\n", rc);
} else {
    printf("✓ Subscrito em %s com sucesso\n", MQTT_SUB_TOPIC);
}

```



```

printf("=== MQTT Loop Principal Iniciado ===\n");

// Loop principal
int count = 0;
while (1) {
    // Verifica status do WiFi periodicamente
    if (count % 60 == 0) { // A cada 60 segundos
        if (!check_wifi_status()) {
            printf("ERRO: WiFi desconectado!\n");
            break;
        }
    }

    // Mantém a conexão viva e processa mensagens recebidas
    if ((rc = MQTTYield(&client, 1000)) != 0) {
        printf("ERRO no MQTTYield: %d\n", rc);
        break;
    }

    // Publica mensagem de teste a cada 10 segundos
    if (++count >= 10) {
        MQTTMessage message;
        char payload[128];

        // Informações do sistema
        uint32_t tick = xTaskGetTickCount();
        int heap_free = xPortGetFreeHeapSize();
        int stack_free = uxTaskGetStackHighWaterMark(NULL) * sizeof(StackType_t);

        snprintf(payload, sizeof(payload),

"{\"device\": \"pico_w\", \"status\": \"online\", \"tick\": %lu, \"heap\": %d, \"stack\": %d}",
            tick, heap_free, stack_free);

        message.qos = QOS0;
        message.retained = 0;
        message.payload = payload;
        message.payloadlen = strlen(payload);

        printf("Publicando em %s:\n%s\n", MQTT_PUB_TOPIC, payload);
        if ((rc = MQTTPublish(&client, MQTT_PUB_TOPIC, &message)) != 0) {
            printf("ERRO ao publicar, código %d\n", rc);
        } else {

```

```

        printf("✓ Mensagem publicada com sucesso\n");
    }
    count = 0;
}

vTaskDelay(pdMS_TO_TICKS(1000));
}

// Cleanup em caso de erro
printf("=== MQTT Task Finalizando ===\n");
MQTTDisconnect(&client);
NetworkDisconnect(&network);
printf("MQTT desconectado. Task finalizada.\n");
vTaskDelete(NULL);
}

```

7.3 Modelo TinyML

- **Coleta de Dados:** 100 amostras por classe (parado, subindo/descendo, esquerda/direita, ziguezague), 1 min cada, 60 Hz.
- **Pré-processamento:**
 - Filtro passa-baixa: Média móvel, frequência de corte 10 Hz.
 - Normalização: Escala [0,1].
 - Janelamento: 1s (60 amostras), 50% sobreposição.
- **Arquitetura:**
 - Entrada: 60 amostras x 3 eixos (180 valores).
 - Camada 1: 32 neurônios, ReLU.
 - Camada 2: 32 neurônios, ReLU.
 - Saída: 4 neurônios, softmax (parado, subindo/descendo, esquerda/direita, ziguezague).
- **Treinamento:** Edge Impulse, validação cruzada, acurácia > 80%.
- **Conversão:** TensorFlow Lite Micro, exportado para C++.
- **Inferência:** Latência < 200 ms no RP2040.

8. Avaliação

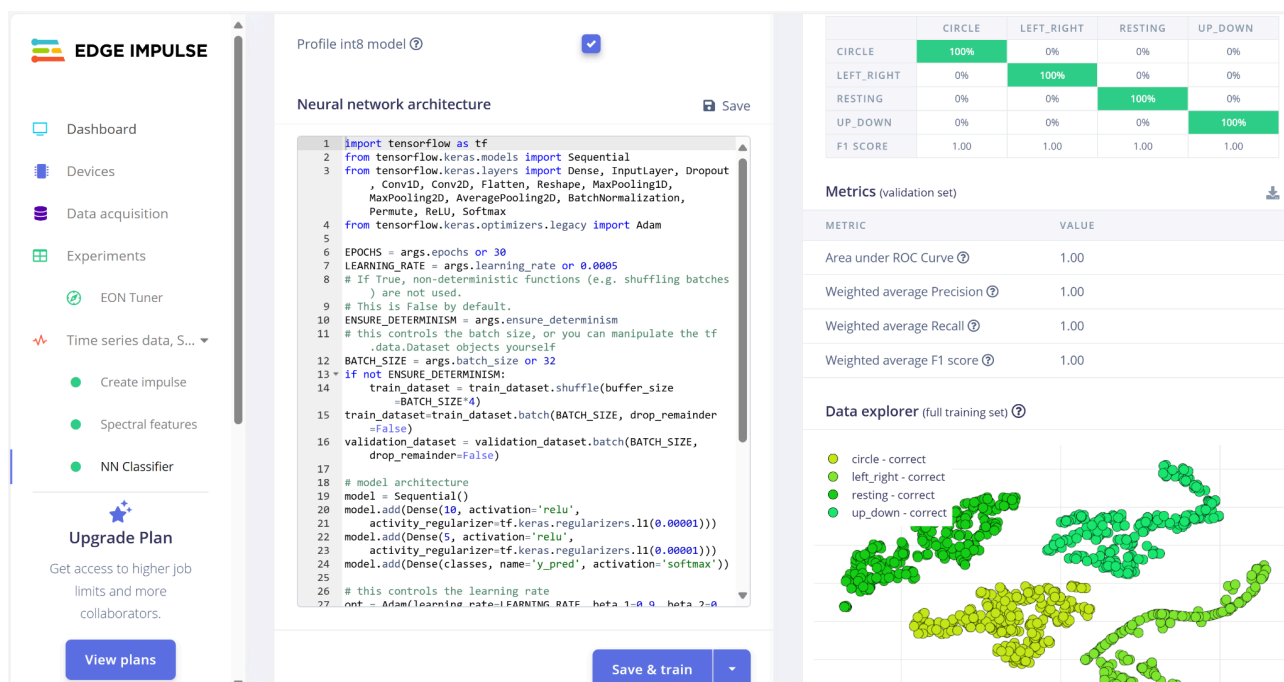
8.1 Critérios

- **Clareza Técnica:** Documentação detalhada com diagramas claros.
- **Qualidade da Documentação:** Código modular, comentado, README no GitHub.
- **Criatividade e Aplicabilidade:** Solução inovadora para logística, escalável.
- **Complexidade e Domínio:** Uso avançado da BitDogLab, TinyML, FreeRTOS, MQTT.
- **Métricas:**
 - Acurácia: > 80%.

- Latência: < 200 ms.
- Autonomia: ≥ 30 dias.
- Estabilidade: 0°C a 50°C.

8.2 Plano de Testes

- **Testes Unitários:**
 - Aquisição de dados: Verificar frequência de 60 Hz.
 - Pré-processamento: Validar filtro e janelamento.
 - Exibição: Confirmar atualização do SSD1306.
 - Transmissão: Testar mensagens MQTT com QoS 2.
- **Testes de Integração:**
 - Verificar interação entre tarefas FreeRTOS.
 - Testar inferência com modelo integrado.
- **Testes de Campo:**
 - Simular movimentos (parado, elevação, deslocamento linear, oscilações).
 - Medir acurácia, latência e consumo.
- **Testes de Estresse:**
 - Operação por 48 horas em 0°C e 50°C.
 - Simular falhas Wi-Fi para verificar reconexão.



9. Cronograma

Etapas	Objetivo	Entregável	Data
1	Definição de Requisitos e Materiais	Documento em Markdown	16/07/2025

2	Arquitetura e Modelagem	Diagramas de hardware e software	04/08/2025
3	Prototipagem e Ajustes	Vídeo/fotos do protótipo, relatório	25/08/2025
4	Entrega Final e Documentação	Sistema funcional, documentação, GitHub	A definir

10. Considerações Finais

O projeto integra sistemas embarcados, TinyML e IoT para resolver um problema crítico na logística de contêineres, oferecendo rastreabilidade detalhada e eficiência energética. A metodologia CUGNASA garante clareza na definição do problema, alinhamento com usuários e uma abordagem estruturada. A solução tem potencial para impactar cadeias logísticas, com aplicações escaláveis em cenários industriais.

Link:

- **vídeo do projeto:** <https://youtu.be/r4KT6auRS9Y?si=LlxuRj-Y9J4jt735>
- **repositório:** [EmbarcaTech-2025/projeto-final-adriana_elias_vagner_tinyml:embarcatech-2025-classroom-projeto-final-projetofinal_template_created_by_GitHub_Classroom](https://github.com/EmbarcaTech-2025/projeto-final-adriana_elias_vagner_tinyml:embarcatech-2025-classroom-projeto-final-projetofinal_template_created_by_GitHub_Classroom)

Glossário

[1]: TinyML: Tiny Machine Learning (TinyML) é um campo emergente da IA e da aprendizagem de máquina (ML) que se concentra no desenvolvimento e implantação de modelos de ML altamente otimizados em dispositivos de hardware com recursos extremamente limitados, como microcontroladores de baixo consumo energético. Esses dispositivos geralmente operam com restrições severas de memória, poder computacional e energia, muitas vezes na ordem de kilobytes de RAM, megahertz de clock e consumo de miliwatts ou menos.

[2]: Edge Impulse: É uma plataforma de desenvolvimento TinyML e machine learning para dispositivos de borda (edge) que permite a criação, treinamento e implantação de modelos de aprendizado de máquina otimizados para microcontroladores (MCUs) e sistemas embarcados de baixo consumo energético.

[3]: AIoT: Artificial Intelligence of Thing é a integração de IA com a IoT, combinando a capacidade de coleta de dados de dispositivos IoT com técnicas avançadas de processamento, como ML e deep learning (DL), para permitir análise em tempo real, tomada de decisão autônoma e otimização de sistemas embarcados.

Tutorial - Edge Impulse

[Tutorials - Edge Impulse Documentation](#)

Referencias Curso UNIFEI-UESTI01- TinyML -

<https://github.com/Mjrovai/UNIFEI-UESTI01-TinyML/tree/main>

Curso Coursera: Edge Impulse - Introdução ao TinyML -

<https://www.coursera.org/learn/introduction-to-embedded-machine-learning>

TinyML - Motion Recognition Using Raspberry Pi Pico -

<https://mjrobot.org/2021/03/12/tinyml-motion-recognition-using-raspberry-pi-pico/>

C. E. Cugnasca, “Projetos de sistemas embarcados”. 2018. Acesso em: 16 de fevereiro de 2025.
[Online]. Disponível em:

<https://integra.univesp.br/courses/2710/pages/texto-base-projetos-de-sistemas-embarcados-%7C-carlos-eduardo-cugnasca>

<https://zlib.pub/book/tinymml-machine-learning-with-tensorflow-lite-on-arduino-and-ultra-low-power-microcontrollers-vshhregc28o0>

https://www.researchgate.net/publication/362806678_Introduction_to_TinyML

<https://upcommons.upc.edu/bitstream/handle/2117/353756/160036.pdf>

<https://digital.futurecom.com.br/artigos/aiot-o-que-e-inteligencia-artificial-das-coisas/>

<https://www.bosch.com.br/noticias-e-historias/aiot/>

<https://www.bing.com/videos/riverview/relatedvideo?q=AloT&mid=7C27CE70B5EADF176AA37C27CE70B5EADF176AA3&FORM=VIRE>

<https://www.bing.com/videos/riverview/relatedvideo?q=AloT&mid=9671E43B51A222DB9F229671E43B51A222DB9F22&FORM=VIRE>

<https://cienciaembarcada.com.br/publicacoes/o-que-e-iiot-e-aiot/>

<https://edgeimpulse.com/>

<https://www.bing.com/videos/riverview/relatedvideo?q=edge+impulse&mid=41C8C11509B60E3C360B41C8C11509B60E3C360B&FORM=VIRE>

<https://aidude.info/pt/ferramentas/edge-impulse>

<https://embarcados.com.br/qualcomm-acquire-edge-impulse/>

