UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Lucas Kendrick Dal Castel

Sistema Embarcado Classificador de Mosquitos Aedes aegypti por Modelo Inteligente

Porto Alegre

2023

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Lucas Kendrick Dal Castel

Sistema Embarcado Classificador de Mosquitos Aedes aegypti por Modelo Inteligente

Projeto de Diplomação II, apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de Engenheiro Eletricista

UFRGS

Orientador: Prof. Dr. José Rodrigo Furlanetto de Azambuja
Coorientador: Prof. Dr. Raphael Martins Brum

Porto Alegre
2023

Lucas Kendrick Dal Castel

Sistema Embarcado Classificador de Mosquitos Aedes aegypti por Modelo Inteligente

Projeto de Diplomação II, apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de Engenheiro Eletricista

BANCA EXAMINADORA

Prof. Dr. Paulo F. Butzen
UFRGS

Prof. Dr. Tiago R. Balen
UFRGS

Prof. Dr. José Rodrigo F. de Azambuja
Orientador - UFRGS

Aprovado em 10 de abril de 2023.

Dedico este trabalho primeiramente a Deus, a quem atribuo minha existência e redenção. Dedico especialmente à minha família, que, com amor, pelo exemplo, ensinou o valor do estudo e do trabalho. Não apenas isso, mas concedeu o suporte necessário para dedicar o tempo para a conclusão desses estudos. Dedico também às amizades, reforçadas ou construídas no caminho. Em especial à turma que me acompanhou desde o ingresso em 2017, colaborando em inúmeros desafios enfrentados.

Agradecimentos

Ao meu orientador, José Rodrigo Furlanetto de Azambuja, pela orientação e minuciosa revisão no decorrer deste trabalho.

Ao meu coorientador, Raphael Martins Brum, pela diligência prestada.

A Eduardo, Luiz e Mateus pelo convite para estudar a vigilância e controle de mosquitos.

A Mariana Recamonde Mendoza e Weverton Cordeiro por manter a linha de pesquisa que resultou no classificador de referência deste trabalho.

De acordo com o Instituto Oswaldo Cruz, o Aedes aegypti foi descrito cientificamente pela primeira vez em 1762, quando foi denominado Culex aegypti. O nome definitivo veio em 1818, após a descrição do gênero Aedes. Em território nacional, desde o início do século 20, o mosquito já era considerado um problema. À época, no entanto, a principal preocupação era a transmissão da febre amarela. “Na campanha contra a febre, o Aedes aegypti foi erradicado do Brasil usando inseticida químico", lembra a pesquisadora [Margareth Capurro].

Porém, não demorou muito para o mosquito voltar e se espalhar pelo extenso território brasileiro. Em meados dos anos de 1980, o Aedes aegypti foi reintroduzido no país, por meio de espécies que vieram principalmente de Cingapura. Hoje, conforme estudiosos, falar em erradicação é algo improvável. “O fato de usarmos muitos inseticidas químicos fez com que sejam selecionados os mosquitos mais resistentes. A resistência atual desses vetores é muito grande. Justamente por isso, tende-se a diminuir ao máximo o uso de inseticida químico”, esclarece Capurro. (OLIVEIRA, 2015)

”Não esmorecer para não desmerecer.”
(Oswaldo Cruz)

Resumo

Doenças transmitidas por vetores são responsáveis por centenas de milhares de mortes todos os anos no mundo todo. O Brasil registrou cerca de 1,5 milhões de casos de dengue em 2019 e cerca de 3 mil casos de microcefalia no surto de zika de 2015/2016, que podem levar a deficiências permanentes, associados ao mosquito Aedes aegypti. O mosquito Aedes aegypti é o maior vetor de dengue e outras arboviroses no país, se reproduzindo em larga escala através do depósito de ovos em recipientes com água parada. Os métodos atuais de vigilância e controle são lentos e insuficientes para reduzir o número de casos no longo prazo. Recentemente, tem-se desenvolvido melhores classificadores de dados, capazes de detectar a espécie de um mosquito através do seu bater de asas. Todavia, isso carece de escalabilidade se dados brutos necessitarem envio contínuo de cada ponto de sondagem para um servidor central de processamento de dados. A escalabilidade pode ser alcançada trazendo o conceito de computação na borda ao se incorporar esses classificadores nas armadilhas que atraem os mosquitos. Isso reduziria drasticamente o volume de trafego de dados exigido. A fim de inserir telemetria nos mecanismos de vigilância de mosquitos já existentes, que dependem de trabalho lento e contínuo para inspeção local dos inúmeros pontos de coleta, este trabalho implementa um classificador de mosquitos Aedes aegypti em um dispositivo de borda. O classificador escolhido já existe em ambiente TensorFlow 2. Sua versão embarcada é implementada usando-se versões personalizadas das bibliotecas livres keras2c e esp32-fft e do framework esp-idf, que é o padrão para o microcontrolador ESP32. Obteve-se saídas as quais foram equivalentes às da implementação original do classificador. Variando configurações e parâmetros do algoritmo, vários tempos de execução foram mensurados. O tempo para executar o classificador, na melhor configuração, foi de 5,21 segundos para um segmento de 1,248 segundos amostrado a 8 kHz. O tempo de execução é suficiente para processamento de intervalos selecionados de maior atividade dos mosquitos.

Palavras-chave: mosquito; classificador; embarcado; CNN; aprendizado de máquina.

Abstract

Vector-borne diseases are responsible for hundreds of thousands of deaths every year worldwide. In Brazil, Dengue recorded about 1.5 million cases in 2019. In the 2015/2016 Zika outbreak, about 3,000 cases of microcephaly that generated permanent disabilities were associated with the mosquito. The Aedes aegypti mosquito is the country's primary vector of dengue and other arboviruses, reproducing on a large scale by laying eggs in standing water. Current surveillance and control methods could be faster and more robust to reduce the number of cases in the long term. Recently better data classifiers have been developed which are capable of detecting the species of a mosquito through its wingbeat. However, this needs more scalability if raw data needs continuous upload from every instrumented spot to a centralized server. Scalability may be achieved by bringing the idea of edge computing by incorporating these classifiers into the traps that attract mosquitoes. That would drasticaly reduce the amount of data traffic required. In order to insert telemetry into the existing mosquito surveillance mechanisms, which currently depends on slow and continuous work for inspection of numerous spots, this work implements an Aedes aegypti mosquito classifier in an edge device. The classifier already exists in a TensorFlow 2 environment. Its embedded version is implemented using custom versions of the free libraries keras2c and esp32-fft with a bug-fixed version of the esp-idf framework, which is the standard for the microcontroller ESP32. Outputs that coincide with the original algorithm's outputs were obtained. By variating configurations and algorithm parameters, many run times were measured. The time to run the classifier in the best setup was 5,21 seconds on the ESP32 from a 1.248 seconds segment sampled at 8 kHz. The run time is sufficient for processing selected intervals of greater mosquitoes activity.

Keywords: mosquito; classifier; CNN; embedded; machine learning.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas

ANN	Artificial Neural Network
API	Application Programming Interface
BLE	Bluetooth Low Energy
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Values
DSP	Digital Signal Processing
DTFT	Discrete-time Fourier Transform
FFT	Fast Fourier Transform
FPU	Floating Point Unit
FT	Fourier Transform
GPIO	General-purpose Input/Output
HAL	Hardware Abstraction Layer
IoT	Internet of Things
MCU	Microcontroller Unit
MLP	Multi-Layer Perceptron
MMU	Memory Management Unit
PSRAM	Pseudo Static Random Access Memory
QSPI	Quad SPI
RAM	Random Access Memory
ReLU	Rectified Linear Unit
ROM	Read-only Memory
RTC	Real-time Clock
SLP	Single-Layer Perceptron
SoC	System-on-chip ou System on a Chip
SPI	Serial Peripheral Interface

SRAM	Static Random Access Memory
STFT	Short-time Fourier Transform
UFRGS	Universidade Federal do Rio Grande do Sul
VBD	Vector-borne Disease
VLSI	Very Large Scale Integration

Sumário

1 Introdução

Doenças transmitidas por vetores (VBDs) são doenças causadas por algum patógeno transmitido por meio de outro organismo (EDER *et al.*, 2018), geralmente um artrópode. Mosquitos (*Culicidae*) e percevejos (*Triatominae*) são exemplos comuns de vetores. Dengue, chikungunya e zika, são doenças virais transmitidas por mosquitos Aedes, enquanto a malária é transmitida por mosquitos *Anopheles*. Assim, percebe-se que controlar o vetor é controlar a proliferação do patógeno (WILSON *et al.*, 2020).

Globalmente, a malária é responsável por cerca de 600 mil mortes anuais, enquanto que a dengue por 40 mil e a febre amarela por 5 mil (OMS, 2014). No Brasil, o vetor de maior preocupação epidemiológica é o Aedes aegypti, responsável por transmitir dengue, chikungunya e zika. Dengue é uma doença endêmica no Brasil e pode levar a complicações fatais, apresentando cerca de centenas de milhares de casos todos os anos. A zika, por outro lado, apresenta menor número de casos, porém levantou altíssima preocupação no surto de 2015/2016, quando uma conexão entre o zikavírus e microcefalia em bebês recém-nascidos foi observada (MLAKAR, 2016; CALVET *et al.*, 2016), com pesquisas posteriormente encontrando ação do vírus em tecido neurológico (GARCEZ *et al.*, 2016).

O mosquito se reproduz depositando ovos sobre água parada (RITA; FREITAS; NOGUEIRA, 2013), sendo o sangue humano importante para a produção dos ovos pelas fêmeas. No país, o combate à dengue se dá com a vigilância das populações de mosquitos através da sua captura em armadilhas espalhadas em espaços urbanos, ilustrado na figura 1. A partir das informações adquiridas, ações diretas e localizadas podem ser tomadas. A aspersão de inseticidas piretróides, bem como o reforço em políticas de conscientização a fim de evitar a formação de focos de criação do mosquito, são mais eficazes quando direcionados aos espaços mais afetados.

Todavia, há algumas dificuldades nesse processo. A captura em armadilhas necessita de um laboroso processo de inspeção individual das armadilhas dispersas do ambiente urbano. Além disso, é esperado um um atraso de até 10 dias entre uma possível captura e o efetivo registro dos dados, pela natureza do processo. Esse tempo é relativamente grande, considerando que o tempo de vida de um mosquito é de ceca de um mês.

Recentemente tem-se desenvolvido mais modelos inteligentes, que classificam as espécies de mosquitos, seja a partir de vídeo, sinais ópticos ou acústicos. Inclusive, já se observa o desenvolvimento de sensores IoT para detecção de mosquitos em campo. Há trabalhos como prova de conceito remetendo dados para outro computador (BOUOTO; FREDDY, 2020), enquanto outros aderem apenas a técnicas clássicas de processamento

Figura 1 – Armadilha para Mosquitos Aedes aegpyti sendo instalada em Porto Alegre

SMS, através da CGVS/EVRV instala armadilhas de monitoramento do mosquito da dengue, no bairro Petrópolis.
Foto: Cristine Rochol/PMPA
Fonte: (SMS PORTO ALEGRE, 2016)

digital de sinais (DSP) que não são capazes de diagnosticar a espécie do mosquito presente (VASCONCELOS *et al.*, 2021).

A informação da espécie é especialmente importante, visto que num mesmo ambiente podem existir populações numerosas de mosquitos inofensivos e de mosquitos potencialmente transmissores de doenças perigosas.

Considerando amostragem de 8 kHz com resolução de 16 bits, cada ponto de sensoreamento produz cerca de 1,3 GB de dados a cada 24 horas. Isso encarece profundamente o processo ao exigir tráfego de grande volume por rede de dados móvel 3G ou 4G, e impossibilita o uso de protocolos de comunicação de banda estreita, comuns em IoT, como LoRaWan e SigFox.

É necessário buscar uma solução com processamento embarcado no instrumento de coleta do sinal sonoro ou optoacústico, a fim de evitar a necessidade de grande tráfego de dados, que colocam em cheque a escalabilidade do método que prevê envio de todos os dados brutos aos servidores em outra localização.

Neste trabalho, busca-se implementar os algoritmos dos classificadores inteligentes num sistema embarcado com restrição de recursos, a fim de permitir sensoreamento remoto de armadilhas de mosquito. Isso pode permitir a otimização do tempo dispendido na inspeção das armadilhas, dado que em vários meses do ano a maioria delas não possui

qualquer mosquito no momento da inspeção. Os resultados permitirão dimensionar melhor as exigências de hardware para aplicações com classificadores similares.

2 Fundamentação teórica

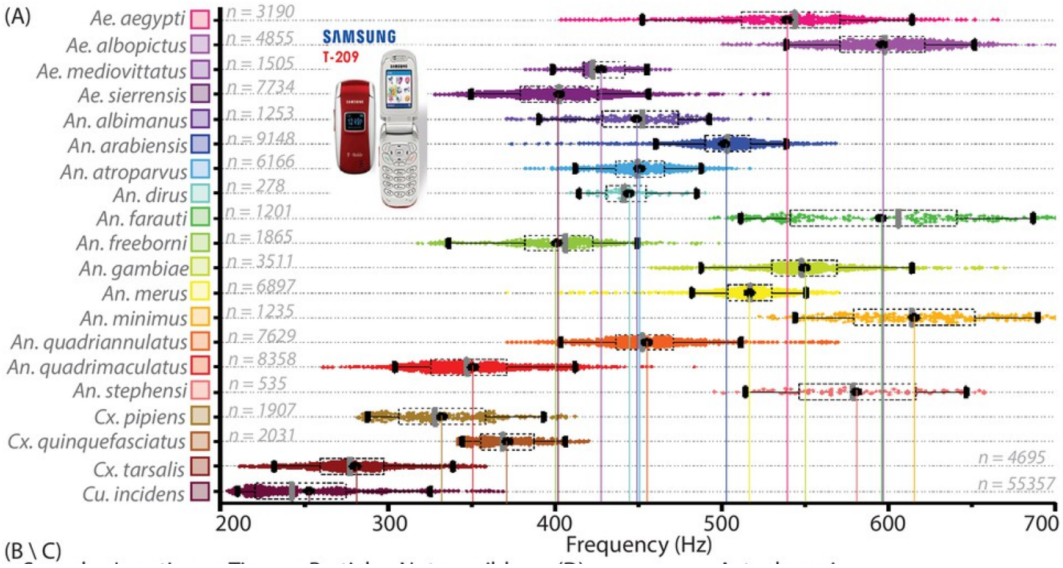
2.1 Doenças transmitidas por vetores

Há várias doenças transmitidas por agentes denominados vetores, que são organismos intermediários portadores do patógeno causador da doença (EDER *et al.*, 2018). Esses vetores eventualmente geram descendentes infectados, formando um estoque selvagem do patógeno, processo que é denominado transmissão vertical. A transmissão vertical permite o transporte do vírus por longas distâncias, além de permitir a existência do vírus em estado inativo (MURILLO; MURILLO; LEE, 2019).

2.2 Mosquitos

Mosquitos são insetos da família *Culicidae*, da ordem dos *Diptera*. Apresentam reprodução sexuada, cujos processos de acasalamento envolvem o sensoreamento do bater de asas da fêmea por parte do macho. A estrutura das antenas dos mosquitos macho são bastante desenvolvidas, a fim de permitir sintonizar as frequências das fêmeas das suas espécies (GIBSON; RUSSELL, 2006). Por conta disso há grande plausibilidade na classificação da espécie e sexo dos mosquitos a partir do sinal do seu bater de asas, seja coletado de forma óptica ou acústica. A figura 2 ilustra a distribuição de frequências fundamentais no conjunto de dados utilizado por espécie.

Figura 2 – Distribuição da frequência fundamental do bater de asas de algumas espécies de mosquito



Fonte: Adaptado de (MUKUNDARAJAN *et al.*, 2017)

2.3 Aprendizado de máquina

Os algoritmos de aprendizado de máquina são extremamente importantes, pois mudam a forma de se resolver um problema usando uma máquina. Da forma clássica, a máquina precisa ser construída ou programada de forma que realize operações pré-determinadas já conhecidas. Entretanto, com aprendizado de máquina, o paradigma é alterado. Agora, informa-se os dados à maquina, e um algoritmo de aprendizagem de máquina muda parâmetros de um modelo que deve se ajustar aos padrões subjacentes. No caso do aprendizado supervisionado, correlacionam-se os dados de entrada com uma saída esperada (LECUN; BENGIO; HINTON, 2015).

Nas subseções seguintes são apresentados alguns tipos de modelos e regras de aprendizagem.

2.3.1 Perceptron

O perceptron é o modelo de neurônio mais básico que deu origem às redes neurais posteriores. No perceptron, há apenas um neurônio com um número arbitrário de entradas e uma saída. A saída desse perceptron é, portanto, uma soma ponderada das entradas com uma função de ativação aplicada à ela.

Um problema que poderia surgir é o ajuste com as entradas zeradas, que seria nulo. Portanto, para permitir valores nulos, tipicamente se adiciona um viés (ou *bias*) ao perceptron, que matematicamente pode ser interpretado como uma entrada sempre com valor 1 e um peso que se chamará *bias*.

A função de saída de um Perceptron é apresentada nas equações 1 e 2.

$$\upsilon(X) = \sum_{k=1}^{k=n}(x_k \cdot w_k) = \begin{bmatrix} w_1 & w_2 & ... & w_n \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} = W \times X, \tag{1}$$

$$y = \phi(\upsilon) \tag{2}$$

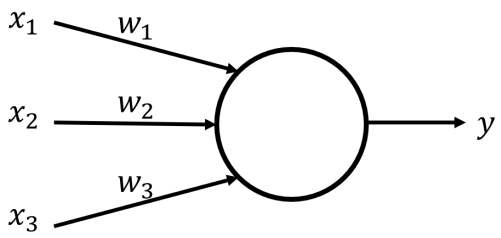
onde

- n é o número de entradas;
- w_1, w_2, ... w_n são os pesos;
- x_1, x_2, ... x_n são os dados de entrada;
- $\upsilon(X)$ é a soma ponderada do perceptron para a entrada X considerando os pesos W;
- ϕ é a função de ativação do perceptron;

- y é a saída prevista;
- W é o vetor linha de pesos do perceptron.
- X é o vetor coluna de entradas;

A Figura 3 mostra os Arquitetura de uma RN com apenas um Perceptron.

Figura 3 – Perceptron



Fonte: o Autor, 2022.

2.3.2 Perceptron de uma camada (SLP)

Modelos *Single-Layer Perceptron* (SLP) são arquiteturas de RNA que só possuem uma camada, a de saída, não possuindo camadas ocultas. Essas RNAs podem apresentar vários perceptrons na saída.

Há uma limitação a respeito da solução de problemas com redes de uma única camada, pois podem apenas resolver problemas linearmente separáveis, limitando muito os problemas passíveis de resolução.

As equações podem ser generalizadas conforme se vê nas equações 3, 4 e 5.

$$W = \begin{bmatrix} W_1 \\ W_2 \\ ... \\ W_i \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & ... & w_{1n} \\ w_{21} & ... & & ... \\ ... & & & ... \\ w_{i1} & w_{i2} & ... & w_{in} \end{bmatrix} \tag{3}$$

$$\upsilon = W \times X = \begin{bmatrix} \upsilon_1 & \upsilon_2 & ... & \upsilon_i \end{bmatrix}^T \tag{4}$$

$$Y = \phi(\upsilon) = \begin{bmatrix} y_1 & y_2 & ... & y_i \end{bmatrix}^T \tag{5}$$

Onde:

- i é o número de neurônios;

- w_{ma} o peso da entrada a para o neurônio m;
- W_m o vetor linha de pesos das entradas do neurônio m;
- υ_m a respectiva soma ponderada;
- y_m a respectiva saída predita;
- W a matriz global de pesos da RNA SLP, de dimensionalidade (i, n);
- υ e y os vetores de somas ponderadas e saídas previstas da RNA SLP.

2.3.3 Perceptron multicamadas (MLP)

Um perceptron multicamadas é uma rede neural artificial com uma ou mais camadas intermediárias que fazem uso de neurônios Perceptrons e é designada MLP (do inglês, *Multi-Layer Perceptron*). Essas redes também podem apresentar vários perceptrons na saída e são necessárias para a solução de problemas não-linearmente separáveis.

(CYBENKO, 1989) mostrou que redes de neurônios (tipo *feedfoward*) com apenas uma camada interna (camada oculta) são capazes de aproximar qualquer função contínua. Essas redes são importantes pois resolvem o problema XOR, percebido desde a década de 1960, que colocou em questionamento a utilidade das redes neurais artificiais (na época de uma única camada), as quais só poderiam ser ajustadas para problemas com um hiperplano de separação.

As *MLPs* usam de equações análogas da *SLP*. Porém, as camadas, com exceção da primeira camada oculta, usam apenas as saídas da camada anterior como entrada no lugar das entradas de dados. Cada camada, portanto, tem uma matriz de pesos e um vetor de vieses (*bias*).

De forma mais genérica, usando-se l como numerador da camada e O_l como o vetor de saída de determinada camada l, as equações poderiam ser expressas para uma *MLP* com $q-1$ camadas ocultas e uma camada de saída da seguinte forma, como se segue nas equações 6 e 7:

$$\upsilon_p = W_p \times O_{p-1} \tag{6}$$

$$O_p = \phi(\upsilon_p) \tag{7}$$

Onde:

- A saída 0 é a saída da camada de entrada (conforme equação 8).

- A saída da última camada é a saída prevista da rede (conforme equação 9).

$$X = O_0 \tag{8}$$

$$Y = O_q \tag{9}$$

2.3.4 Aprendizado Supervisionado de uma Rede Neural Artificial (RNA)

De forma geral, um algoritmo de aprendizado supervisionado compara a saída predita pelo modelo atual e a saída esperada e iterativamente ajusta os parâmetros de acordo com o erro e as entradas.

1. inicializar os pesos da rede neural artificial (RNA) com valores adequados;
2. disponibilizar na camada de entrada (camada que não realiza processamento) da RNA os dados de treinamento;
3. determinar a saída da RNA e calcular o erro da saída predita em relação a saída desejada;
4. ajustar os pesos e reduzir o erro;
5. repetir os passos 2 a 3 para todos os dados do conjunto de treinamento.

O algoritmo que modifica os pesos é denominado a Regra de Aprendizagem.

2.3.4.1 Regra Delta

Uma regra importante de aprendizagem é a regra delta, que é definida para o aprendizado da SLP.

A Regra Delta (Regra Adaline ou de Widrow-Hoff) ajusta os pesos (com o *bias*) da RNA de acordo com o seguinte algoritmo: "Se um nó de entrada contribui para o erro do nó de saída, o peso entre os dois nós é ajustado proporcionalmente ao valor de entrada, x_j e o erro de saída, e_i". A Equação 10 exibe essa regra.

$$'w_{ij} \leftarrow w_{ij} + \underbrace{\alpha\ e_i\ x_j}_{\Delta w}, \tag{10}$$

onde:

- w_{ij} é o peso entre o nó de saída i e nó de entrada j;
- Δw é a variação do peso entre duas épocas sucessivas;

- α é a Taxa de Aprendizagem ($0 < \alpha \leq 1$);
- e_i é o erro do nó de saída i;
- x_j é a saída do nó da entrada j, ($j = 1, 2, 3$);

2.3.4.2 Generalização da Regra Delta

A regra Delta pode ser modificada para a utilização de qualquer função de ativação arbitrária, considerando a Equação 11:

$$\delta_i = \phi^{'}(v_i)\ e_i, \tag{11}$$

onde:

- e_i é o erro do nó de saída i;
- v_i é a soma dos pesos do nó de saída i;
- $\phi^{'}$ é a derivada da função de ativação do nó de saída i;

Dessa forma, a Equação 10 pode ser escrita de forma geral como 12.

$$w_{ij} \leftarrow w_{ij} + \underbrace{\alpha\ \delta_i\ x_j}_{\Delta w}, \tag{12}$$

onde:

- w_{ij} é o peso entre o nó de saída i e nó de entrada j;
- Δw é a variação do peso entre duas épocas sucessivas;
- α é a Taxa de Aprendizagem ($0 < \alpha \leq 1$);
- δ_i é a multiplicação da derivada da função de ativação pelo erro do nó de saída i;
- x_j é a saída do nó da entrada j, ($j = 1, 2, 3$);

2.3.4.3 Algoritmo de retro-propagação - *back-propagation algorithm*

A regra delta é ineficiente para treinamento de redes de múltiplas camadas. Isso se dá especialmente pois não há erro conhecido para as saídas das camadas ocultas. Entretanto, (RUMELHART; HINTON; WILLIAMS, 1986) propuseram o algoritmo de retro-propagação. A grande diferença dos outros algoritmos é que um erro estimado é fornecido aos neurônios das camadas ocultas, o que permite aplicar a regra delta modificada tomando os erros estimados para cada camada oculta.

O algoritmo nada mais é do que a aplicação da regra da cadeia de trás pra frente (LECUN; BENGIO; HINTON, 2015). Dessa forma, o algoritmo de retro-propagação pode ser resumido da seguinte forma:

$$e_{k-1} = W_k^T \times \delta_k \tag{13}$$

Sendo k o número da última camada, e o erro estimado, W a matriz de pesos de entrada da camada e δ o delta da regra delta.

2.4 Sistemas embarcados

Sistemas embarcados são sistemas no qual há um computador dedicado à sua operação, tornando-se assim um computador de uso específico. Geralmente um sistema embarcado precisa de menos recursos computacionais que os oferecidos por um computador de uso geral, comum em ambientes de trabalho e computadores pessoais.

2.4.1 Microcontrolador

Um microcontrolador é um microprocessador integrado aos periféricos necessários para o seu uso como um computador. Há vantagens de se reduzir o custo unitário do sistema com VLSI (do inglês, Very Large Scale Integration), sendo que muitos microcontroladores são produzidos para aplicações de baixo processamento, custo e preço.

São usados extensivamente em sistemas embarcados, pois viabilizam a computadorização de aplicações menores. No objeto de estudo desde trabalho, o sistema embarcado que se pretende sensorear são as armadilhas de insetos. É desejado que um sistema de tal tipo seja economicamente viável, melhorando a alocação de recursos e para isso procura-se uma solução de baixo custo.

Todavia, tais microcontroladores de baixo custo e baixo consumo tendem a ter menos recursos disponíveis. Desenvolver um sistema com limitação de recursos exige muitas vezes otimização do algoritmo para determinado hardware, com o fim de extrair o máximo de poder dele.

Quando há interesse em se utilizar uma quantidade de memória maior que a capacidade de endereçamento do processador, a técnica denominada *bank switching* pode ser empregada. Funciona modificando o mapeamento entre os endereços do processador e as regiões da memória física em tempo de execução. Dessa forma, pode-se permitir o acesso de regiões diferentes da memória em diferentes momentos. Um exemplo de uso desse mecanismo está na linha ESP32 Series quando se utiliza memória RAM externa de tamanho maior que 4 MB, como está detalhado na seção 2.5.5.

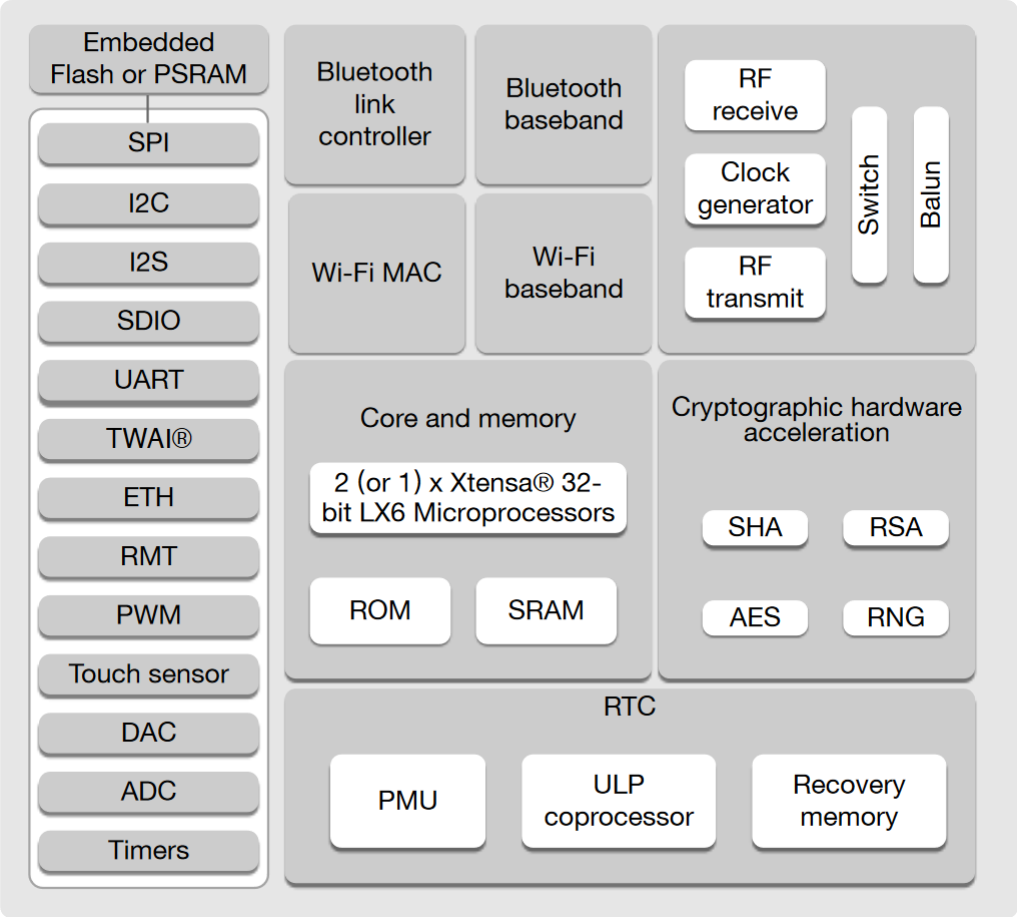
2.5 ESP32

ESP32 é um "Wi-Fi-and-Bluetooth combo chip"desenvolvido com tecnologia TSMC low-power 40nm pela Espressif Systems. O ESP32 possui um ou dois processadores Xtensa LX6, apesar dos dispositivos usados neste trabalho terem sido todos dual-core.

O Xtensa LX6 tem pipeline de sete estágios, o que permite atingir frequência de relógio de até 240 MHz (porém 160 MHz para o ESP32-S0WD) e segundo o fabricante tem FPU. O conjunto de instruções é de 16/24-bit, o que segundo o fabricante permite obter alta densidade de código.

O framework oficial de desenvolvimento é o esp-idf (ESPRESSIF, 2023a).

Figura 4 – Diagrama de blocos funcionais do ESP32

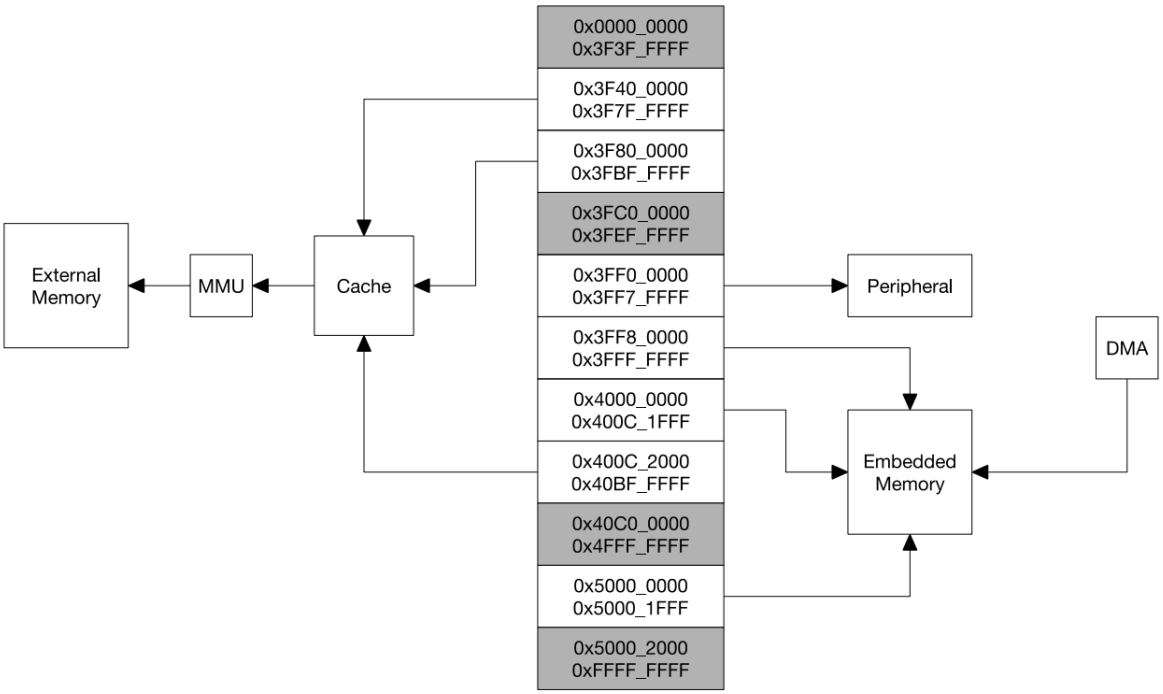


Fonte: Adaptado de (ESPRESSIF, 2022)

2.5.1 CPU e memória:

- Microprocessador Xtensa single/dual-core 32-bit LX6
- 448 KB ROM
- 520 KM SRAM

Figura 5 – Arquitetura e mapeamento de memória no ESP32



Fonte: Adaptado de (ESPRESSIF, 2022)

- 16 KB SRAM no RTC
- Suporte a QSPI para chips flash e SRAM

Tabela 1 – Mapeamento de Memória no ESP32

Category	Target	Start Address	End Address	Size
Embedded Memory	Internal ROM 0	0x4000_0000	0x4005_FFFF	384 KB
	Internal ROM 1	0x3FF9_0000	0x3FF9_FFFF	64 KB
	Internal SRAM 0	0x4007_0000	0x4009_FFFF	192 KB
	Internal SRAM 1	0x3FFE_0000	0x3FFF_FFFF	128 KB
		0x400A_0000	0x400B_FFFF	
	Internal SRAM 2	0x3FFA_E000	0x3FFD_FFFF	200 KB
	RTC FAST Memory	0x3FF8_0000	0x3FF8_1FFF	8 KB
		0x400C_0000	0x400C_1FFF	
	RTC SLOW Memory	0x5000_0000	0x5000_1FFF	8 KB
External Memory	External Flash	0x3F40_0000	0x3F7F_FFFF	4 MB
		0x400C_2000	0x40BF_FFFF	11 MB+248 KB
	External RAM	0x3F80_0000	0x3FBF_FFFF	4 MB

Fonte: Adaptado de (ESPRESSIF, 2022)

2.5.2 Temporizadores e osciladores de relógio:

- Oscilador interno de 8 MHz com calibração

- Oscilador interno RC com calibração
- Suporte a oscilador externo a cristal na faixa de 2 MHz a 60 MHz (requer 40 MHz para Wi-Fi e Bluetooth)
- Suporte a cristal oscilador externo de 32 kHz para RTC com calibração
- Dois grupos de temporizadores, incluindo 2 temporizadores de 64 bits e um watchdog principal em cada grupo
- Temporizador RTC
- *Watchdog* RTC

2.5.3 Interfaces de periféricos avançadas:

- 34 x GPIOs
- ADC SAR de 12 bits de até 18 canais
- 2 x DAC de 8 bits
- 10 x sensores de toque
- 4 x SPI
- 2 x I2S
- 2 x I2C
- 3 x UART
- 1 host (SD ou eMMC ou SDIO)
- 1 slave (SDIO ou SPI)
- Ethernet MAC interface com DMA dedicado e suporte a IEEE 1588
- TWAI compatível com ISO 11898-1 (Especificação CAM 2.0)
- RMT (TX/RX)
- PWM para motor
- PWM para LED de até 16 canais

2.5.4 Capacidades de Segurança:

- Boot seguro
- Flash criptografado
- One-time pad de 1024 bits, de até 768 bits para consumidores
- Aceleração de hardware para criptografia:

- AES
- Hash (SHA-2)
- RSA
- ECC
- Random Number Generator (RNG)

2.5.5 Bank switching no ESP32

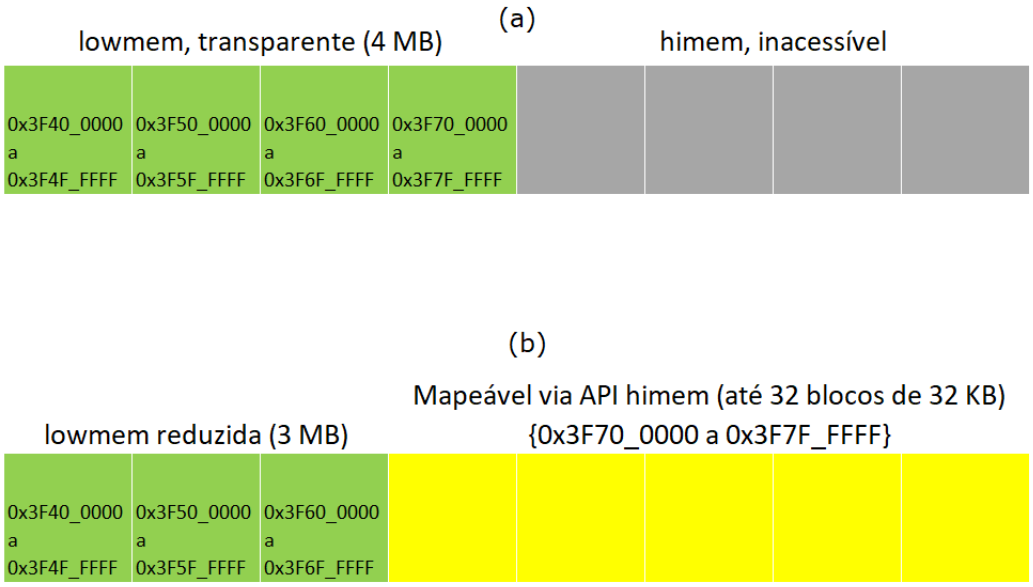
Na figura 5 e na tabela 1, tem-se os endereços reservados para cada tipo de memória. Como se percebe, há apenas 4 MB de endereços reservados para memória RAM externa, o que vai de encontro à disponibilidade de módulos ESP32 vendidos com chips de memória RAM externa de 8 MB. Isso se dá pois no projeto do ESP32 considerou-se a possibilidade de *bank switching*. O componente esp-psram do *framework* esp-idf provê a API himem, que é responsável por fazer o *bank switching*.

A API himem reserva entre 1 a 62 (inclusive) blocos de 32 KB (equivalentes ao cache de memória) ao fim dos endereços reservado para memória externa. Esses blocos, que antes poderiam estar sob posse do alocador padrão, se configurado para isso, que recebe chamadas via método malloc(...) ou heap_caps_malloc(...), passam a estar disponíveis apenas sob a API himem, porém com a vantagem de guardar um volume maior de dados.

A figura 6 mostra um caso de uso da API himem, no qual há 8 MB de RAM externa no ESP32 (na versão v5.0 no esp-idf apenas 8 MB de RAM externa é suportada pela API himem). O número de blocos reservados é de 32, totalizando um total de $32 \cdot 32$ kB = 1 MB.

A API himem, no entanto, tem algumas limitações. A alocação e o mapeamento se dão em handles, exigindo a rotina que faz uso de dados armazenados na região controlada pela API himem conhecer essa API. Isso pois, mesmo havendo possibilidade de abstração, o tempo necessário para mapear e desmapear regiões específicas pode ser elevado, especialmente quando se pretende fazer acesso não sequencial. Além disso, só permite alocar e mapear em blocos cujo tamanho é um múltiplo de 32 kB. Logo, um vetor menor, ou deve ser aglutinado em outro maior, ou simplesmente ocupará um bloco inteiro de 32 kB.

Figura 6 – Exemplo de bank switching com 8 MB de RAM externa



(a) Sem uso da API himem, apenas metade da memória está acessível. (b) Ao se reservar 32 blocos (de 32 kB), apenas 3 MB de memória podem ser acessados diretamente, porém há outros 5 MB disponíveis.
Fonte: o Autor, 2023.

2.6 Representações de Fourier

As representações de Fourier consistem nas decomposições de sinais em uma superposição ponderada de senoides complexas (ou a equivalente soma de senoides e cossenoides reais). A partir dessas técnicas, temos a representação do sinal que era conhecido no domínio do tempo, agora, no domínio da frequência.

Essa representação é muito útil em várias esferas do conhecimento, no qual um sinal de alta complexidade pode ter a disposição de sua informação simplificada nessa nova representação. O que para análises de sinais sonoros e de radiofrequência é uma técnica praticamente ubíqua. Assim, o timbre acústico de determinado instrumento musical – ou inseto – pode ter suas faixas de frequências observadas bem como as proporções entre as componentes espectrais.

Há quatro representações, sendo cada qual definida em termos de duas qualidades do sinal a ser transformado: 1) Série de Fourier, para sinais contínuos e periódicos; 2) Série de Fourier, para sinais discretos e periódicos; 3) Transformada de Fourier, para sinais contínuos e não periódicos; 4) Transformada de Fourier de Tempo Discreto, para sinais discretos e não periódicos.

Tratando-se o objeto deste trabalho um sistema digital, todos seus sinais são amostrados e, portanto, discretos no tempo. Assim, nas equações 14 e 15, tem-se respectivamente

as definições da Série de Fourier de Tempo Discreto e da Transformada de Fourier de Tempo Discreto segundo (HAYKIN; VEEN, 2002).

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\mathrm{j}k\Omega_0 n}, \qquad \Omega_0 = 2\pi/N \tag{14}$$

$$X(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x[n] \mathrm{e}^{-\mathrm{j}\Omega n} \tag{15}$$

2.6.1 Transforamda rápida de Fourier (FFT)

A FFT (do inglês, Fast Fourier Transform) é um algoritmo da transformada de Fourier discreta que apresenta uma complexidade computacional menor e por conta disso é amplamente usada para processamento digital de sinais. A implementação literal da DTFT geralmente leva a um algoritmo cuja complexidade é O(N^2), tornando-a altamente custosa a medida que N aumenta. Entretanto, com o algoritmo de (COOLEY; TUKEY, 1965), essa complexidade é reduzida para O(N log(N)).

2.6.2 Transformada de Fourier de curto termo (STFT)

A STFT (do inglês, Short-Time Fourier Transform) é uma operação que produz um conjunto de transformadas de Fourier sobre um o sinal multiplicado por uma janela de análise. A equação 17 define a STFT de acordo com (PORTNOFF, 1980). Também pode se representar na equação 19 em termos da sequência de curto termo definida em 18. A figura 7 ilustra esse conceito.

$$Y = \mathrm{STFT}\{x\} \tag{16}$$

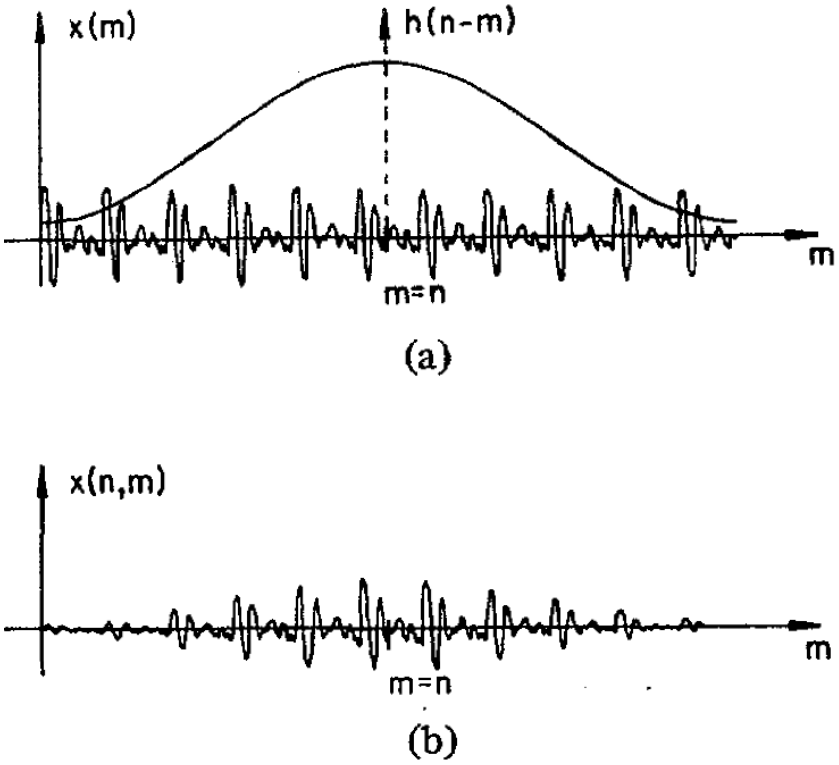
$$\mathrm{Y}(n, \omega) = \sum_{m=-\infty}^{\infty} h(n-m)x(m)\mathrm{e}^{-j\omega m} \tag{17}$$

$$x(n, m) = h(n-m)x(m) \tag{18}$$

$$\mathrm{Y}(n, \omega) = \sum_{m=-\infty}^{\infty} x(n, m)\mathrm{e}^{-j\omega m} \tag{19}$$

Figura 7 – Sequência de curto termo

(a) Janela de análise deslocada e invertida no tempo h(n-m) superposta aos dados x(m). (b) Sequência de curto tempo x(n,m) = h(n-m)x(m) para um valor arbitrado de n.



Fonte: Adaptado de (PORTNOFF, 1980)

Na seção 2.8, há maiores detalhes sobre a janela de análise h(n-m). Por hora, considere a janela retangular como um pulso de amplitude 1 e largura L centrado em n. Essa janela é responsável por selecionar qual parte do sinal será considerado ao calcular a transformada. Nesse caso, esse somatório é exatamente equivalente a:

$$\begin{cases} x(n,m) = x(m) & | \quad (n-m) \in [-(L/2 - 1), L/2] \\ x(n,m) = 0 & | \quad (n-m) \notin [-(L/2 - 1), L/2] \end{cases} \tag{20}$$

Dessa forma, a expressão da STFT fica conforme a equação 21:

$$\mathrm{Y}(n,\omega) = \sum_{m=n-L/2}^{n+(L/2-1)} x(m)\mathrm{e}^{-j\omega m} \tag{21}$$

Enquanto o segundo caso da eq. 20 for o mesmo em outras janelas, os limites do somatório poderão ser mantidos. Seguindo as premissas da eq. 20, temos a eq. 23 a partir da eq. 22.

$$k = m - n \tag{22}$$

$$\mathrm{Y}(n,\omega) = \left[\sum_{k=-L/2}^{(L/2-1)} x(n+k)\mathrm{e}^{-j\omega k}\right]\mathrm{e}^{-j\omega n} \tag{23}$$

Generalizando, tem-se a STFT na eq. 24 para o sinal janelado:

$$\mathrm{Y}(n,\omega) = \left[\sum_{k=-L/2}^{(L/2-1)} h(-k)x(n+k)\mathrm{e}^{-j\omega k}\right]\mathrm{e}^{-j\omega n} \quad , \quad h(-k) = 0 \mid k \notin [-(L/2-1), L/2] \tag{24}$$

Seja N_{total} o número de amostras capturadas no segmento analisado. No intervalo $[0; N_{total}-1]$ estão as amostras do sinal capturado. Entretanto, percebe-se que o domínio da STFT será, portanto, apenas $[L/2; (N_{total}-L/2)]$. Para o domínio da STFT ser equivalente ao domínio do vetor $\mathbf{x}$, o somatório exige que $\mathbf{x}$ tenha o domínio $[-L/2; (N_{total}-1)+(L/2-1)]$.

Isso significa que o vetor precisa ser preenchido com $L/2$ amostras no seu início (em $[-L/2; -1]$) e $(L/2-1)$ no seu final (em $[N_{max}; (N_{max}-1)+(L/2-1)]$). Para satisfazer essa premissa, utiliza-se de preenchimento (ou *padding*), que é melhor discutido na seção 2.7.

Destaca-se que o preenchimento nas bibliotecas librosa é simétrico, adicionando-se $L/2$ em ambas as extremidades do vetor de entrada. Isso implica que o domínio da STFT em relação a n é $[0; N_{total}]$, e não $[0; (N_{total}-1)]$ como no sinal de entrada.

2.6.3 Espectrograma

De acordo com (ALMEIDA, 1994), o espectrograma de um sinal é usualmente definido como os quadrados dos módulos dos elementos da STFT, representando a energia em cada faixa espectral. Dessa forma, o termo relativo à fase ($\mathrm{e}^{-j\omega n}$) da eq. 24 não precisa ser calculado para obter-se o espectrograma.

2.6.3.1 Gerando Espectrograma a partir da FFT

Seja $\mathbf{S}$ o espectrograma de $\mathbf{x}$

$$\mathbf{S} := \mathrm{ESPECTROGRAMA}\{\mathbf{x}\} \tag{25}$$

O espectrograma pode ser definido como os quadrados das magnitudes da STFT, conforme eq. 26:

$$\mathbf{S}(n,\omega) = \left|\sum_{k=-L/2}^{(L/2-1)} h(-k)x(n+k)\mathrm{e}^{-j\omega k}\right|^2 \quad , \quad h(-k) = 0 \mid k \notin [-L/2, (L/2-1)] \tag{26}$$

Percebe-se que o espectrograma pode ser definido em termos da DTFT. Para fazer isso, a entrada da DTFT será a convolução da janela com o sinal preenchido e deslocado, conforme eq. 27. Onde $\mathbf{x_{FFT}}$ é o sinal que será a entrada da FFT.

$$\mathbf{x_{FFT}}(\mathrm{k}) = h(-k)x(n+k) \tag{27}$$

Pode-se construir um espectrograma baseado na FFT desses segmentos na eq. 28:

$$\mathbf{S}(n,\omega) = |\mathrm{FFT}\{x_{FFT}(n,k)\}|^2 \quad , \quad h(-k) = 0 \mid k \notin [-{}^{NFFT}/_2, ({}^{NFFT}/_2 - 1)] \tag{28}$$

2.6.3.2 Comprimento de Salto ou *Hop Length*

Todavia, calcular uma FFT para cada elemento existente no sinal nem sempre é de interesse, pois apesar do maior detalhamento, praticamente os mesmos dados são usados para calcular o espectrograma várias vezes. Um espectrograma mais detalhado tem custo computacional maior, muitas vezes sem trazer ganhos suficientes que justifiquem a sua adoção. Uma maneira de gerar um espectrograma de forma menos custosa é usando-se de saltos maiores que um elemento.

O espectrograma baseado em saltos pode ser definido de forma geral na eq. 29:

$$\mathbf{S_{saltos}}(i,\omega) := \mathbf{S}(i \cdot hop\_length, \omega) \tag{29}$$

Assim, tem-se uma resolução temporal no espectrograma que é $\frac{sample\_rate}{hop\_length}$. Onde $sample\_rate$ é a taxa de amostragem e hop_length é o comprimento do salto em amostras.

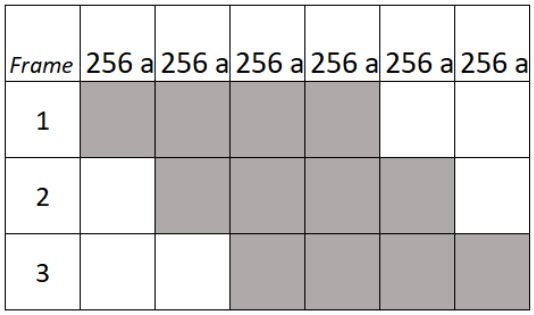
2.7 Preenchimento ou *padding*

Usualmente em DSP se utilizam FFTs em potências de 2 ou 4 (2-radix e 4-radix). Isso de dá para aumentar a eficiência em harwdare otimizado para tal ou simplesmente tornar possível que determinado software performe a operação. Desse fato, decorre que, eventualmente, há a necessidade de preenchimento do vetor de dados a fim do seu tamanho corresponder a um valor que seja potência de 2. Esse procedimento é denominado preenchimento (ou *padding*) e pode ser realizado com algum de vários tipos. Para DSP, um caso comum é o *Zero Padding*, que consiste em preencher com valores nulos no começo e/ou no fim do sinal (DONNELLE; RUST, 2005).

O preenchimento existe no contexto da STFT também, de forma a permitir que determinado intervalo do sinal seja avaliado de forma a permitir manter o número de quadros (ou *frames*) independente do número da STFT.

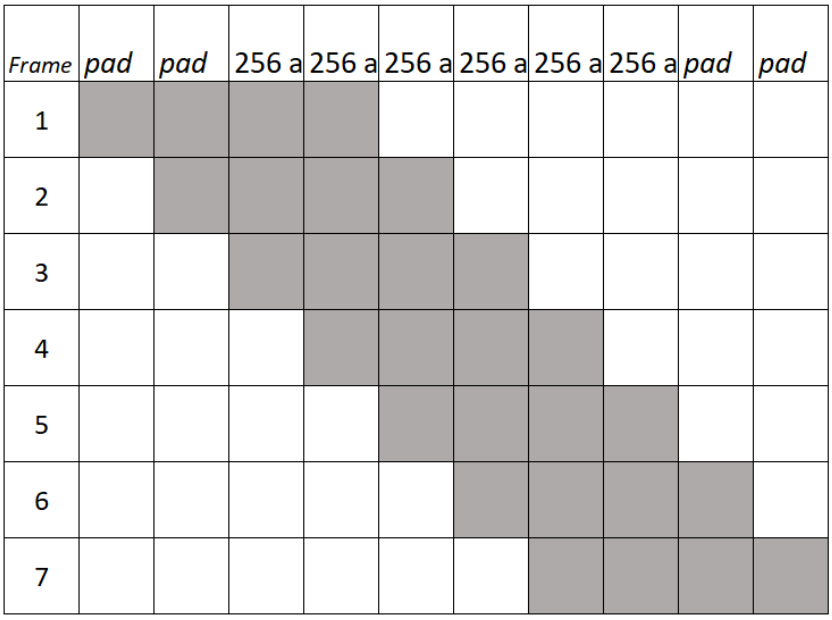
As figuras 8 e 9 exemplificam quais dados estariam presentes em cada quadro produzido no contexto de uma STFT. Neste exemplo, o período da FFT é 1024, o comprimento de salto é 256. O primeiro caso não usa de preenchimento enquanto o segundo usa de preenchimento de 1024 amostras distribuídas metade em cada extremidades. O segundo exemplo usa um segmento de 1536 amostras do sinal para fins de exemplificação. Percebe-se que, sem preenchimento, o número de quadros produzidos é de apenas 3, enquanto o comprimento do salto é de apenas 1/6 do número de amostras no segmento. Por outro lado, com preenchimento, temos 7 quadros encontrados.

Figura 8 – Exemplo de processamento de STFT sem padding



Fonte: o Autor, 2023.

Figura 9 – Exemplo de processamento de STFT com padding



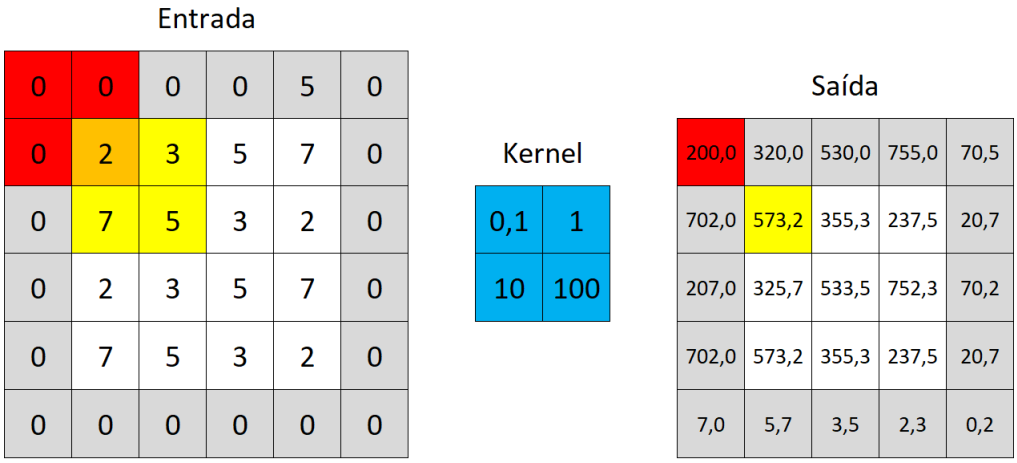
Fonte: o Autor, 2023.

Em camadas convolucionais de redes neurais, também pode-se usar a técnica de preenchimento a fim de permitir o uso de um *kernel* de tamanho arbitrário, sem necessariamente reduzir as dimensões da camada de saída.

Na figura 10 mostra-se um exemplo de uso de preenchimento em uma camada de tipo Conv2D, com *stride* = 1, *padding* = 1, *bias* nulo, entrada de formato 4x4 e saída 5x5. Observe que sem o preenchimento ativado, a saída seria apenas formada pelos elementos coloridos em amarelo e branco, o que resultaria no formato de saída 3x3.

A figura usa cores vermelho e amarelo para demonstrar quais regiões foram usadas como entrada para dois casos de elementos da saída. Em vermelho está um exemplo de elemento acrescentado à saída devido a presença do preenchimento, e em amarelo o primeiro elemento caso o preenchimento não fosse adicionado. Coloridos em cinza na entrada e na saída estão as regiões acrescentadas na camada de entrada e que por conseguinte geraram a região cinza na camada de saída.

Figura 10 – Exemplo de Conv2D com padding e *bias*



Fonte: o Autor, 2023.

2.8 Janela de análise ou *window*

Ao se coletar um segmento de um sinal arbitrário periódico, não raro, o comprimento do segmento não coincide com o período do sinal. Isso produz uma sinal truncado com características espectrais diferentes, chamado de vazamento espectral. Ajustes podem ser feitos no sinal de entrada com diferentes funções de janelamento (ou *windowing*). Maiores detalhes são abordados por (HARRIS, 1978).

Pode-se abstrair a não aplicação de uma janela modificadora à janela retangular, que é constante e unitária em todo o segmento coletado e nula no restante da amostra. Isso pode ser representado pela equação 30:

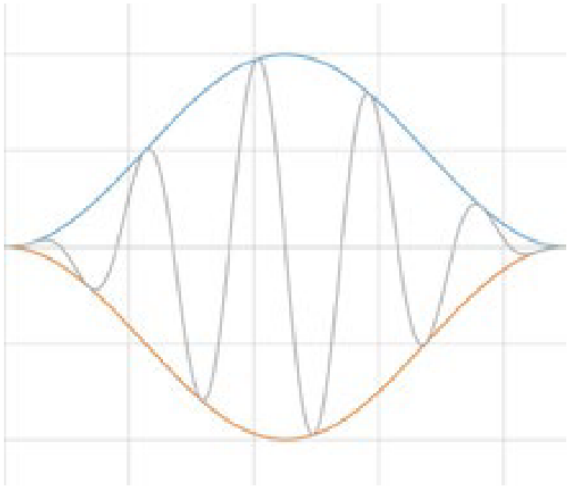
$$\begin{aligned} w[n] &= 1 \\ n &= 0, 1, ..., N-1 \end{aligned} \tag{30}$$

Uma janela que tradicionalmente reduz o vazamento espectral (NATIONAL INSTRUMENTS, 2021) é a janela de hann, nomeada em honra ao climatologista Julius von Hann. A equação 31 descreve essa janela, onde $n = 0$ é o centro do segmento. Por outro lado, a equação 32 é alinhada com o sinal onde $n = 0$ coincide com a extremidade inicial do segmento. Percebe-se que, apesar de exigir cálculo de cossenos, o custo computacional permanece baixo ao manter armazenada uma tabela com os valores usados pois independe do sinal de entrada. A figura 11 retrata visualmente o resultado da aplicação da *hann* em um sinal senoidal arbitrário. Percebe-se que a descontinuidade nas extremidades do recorte é suprimida pois a hann é quase nula nas extremidades da janela.

$$\begin{aligned} w[n] &= 0.5 + 0.5 \cdot cos(\frac{2n}{N} \cdot \pi) \\ n &= -\frac{N}{2}, ..., -1, 0, 1, ..., \frac{N}{2} - 1 \end{aligned} \tag{31}$$

$$\begin{aligned} w[n] &= 0.5 - 0.5 \cdot cos(\frac{2n}{N} \cdot \pi) \\ n &= 0, 1, ..., N - 1 \end{aligned} \tag{32}$$

Figura 11 – Aplicação da *Janela de Hann (ou hanning)* em uma senoide



Exemplo de Janela de hann (em azul), aplicada a um sinal senoidal (em cinza). E escala vertical equivalente a uma unidade por divisão.
Fonte: (NATIONAL INSTRUMENTS, 2021)

2.8.1 Escala Mel

A escala Mel consiste na ideia de se criar uma escala de frequências subjetivamente percebidas pelo ouvido e mente humana. A primeira publicação relacionada, de 1937, fornece uma tabela de resultados experimentais subjetivos e um esboço da escala num

gráfico (STEVENS; VOLKMANN; NEWMAN, 1937). Desde então, o precedente para mais subjetividades entrarem e várias escalas surgirem com o mesmo conceito estava firmado. Em 2000, O'Shaughnessy, em seu livro, define a escala mel com a fórmula da eq. 33.

$$mel(f) = 2595 \cdot log_{10}(1 + \frac{f}{700}) \tag{33}$$

Entretanto, essa não é a fórmula padrão utilizada pela biblioteca librosa; a padrão é a denominada Slaney, que seria a mesma do Auditory Toolbox para Matlab do Slaney (SLANEY, 1998). A função librosa.filters.mel(...) ainda permite usar a fórmula da equação 33 sob o nome do Hidden Markov Model Toolkit (UNIVERSITY OF CAMBRIDGE, 2016).

Internamente, na librosa, a conversão padrão é definida pela equação 34:

$$\begin{aligned} mel(f) &= \frac{f}{200/3} \mid f \leq 1000Hz \\ mel(f) &= \frac{1000}{200/3} + \frac{log_2(\frac{f}{1000})}{log_2(6.4)/27} \mid f > 1000Hz \end{aligned} \tag{34}$$

A biblioteca librosa, internamente, ao chamar-se o método melspectrogram, gera um espectrograma convencional, como descrito na seção 2.6.3. A partir desse, uma função converte para o espectrograma em escala mel, ponderando proporcionalmente de acordo com a proximidade de cada frequência com as bandas mel adjacentes.

2.9 Plataformas e bibliotecas para desenvolvimento de modelos inteligentes

2.9.1 TensorFlow

TensorFlow (GOOGLE, 2022b) é uma plataforma que permite criar com facilidade modelos de aprendizagem de máquina, treiná-los e executá-los. Possui a API Keras, que permite criar modelos inteligentes em alto nível, abstraindo por tipos de camadas de uso comum e dispensando a especificação individual de cada neurônio e suas interconexões.

2.9.2 TensorFlow Lite

É a biblioteca móvel da plataforma TensorFlow que permite executar os modelos inteligentes, previamente treinados, em dispositivos embarcados como telefones celular e microcontroladores (GOOGLE, 2022c). Muitos modelos TensorFlow podem ser convertidos para TensorFlow Lite sem muitos empecilhos, porém o número de capacidades e tipos camadas existentes é restrito.

A vantagem frente a outras opções é a vasta base de usuários, bem como ter suporte rápido para conversão de modelos com parâmetros em float32 para uma versão quantizada em int8. A biblioteca pode ser compilada parcialmente, inclusive possuindo versão para microcontroladores, de 16KB. Porém apresenta alto nível de abstração, dificultando mudanças pontuais.

2.9.3 Keras2C

É uma biblioteca em C e um script em Python que permitem converter um modelo TensorFlow construído sob a API Keras para código em C puro (CONLIN, 2022).

Observa-se que há limitações relacionadas à implementação de algumas camadas Keras não estar completa, dificultando o uso de alguns modelos inteligentes que usem camadas de alguns tipos específicos, como a LayerNormalization, por exemplo (CONLIN, 2022). Há uma base de usuários menor e não foi extensivamente testada, tendo o Autor sugerido correções enquanto testava suas funcionalidades (DAL CASTEL, 2023). A vantagem está na simplicidade da biblioteca, sendo flexível para otimizações especificas, o que a torna a extremamente útil para este trabalho.

3 Desenvolvimento

Neste capítulo as fases principais de desenvolvimento do sistema e a justificativo por trás de algumas decisões de projeto são descritas. O software desenvolvido será disponibilizado no repositório <https://github.com/lucaskdc/ESP32BinaryMosquitoClassifier> na plataforma GitHub.

3.1 Escolha da infraestrutura

Ao se desenvolver sistemas embarcados, é comum encontrar ferramentas de desenvolvimento, HALs e frameworks específicos em cada plataforma. Existem ferramentas e bibliotecas que aumentam o nível de abstração, como PlatformIO, com o fim de expandir a portabilidade do software escrito sob essas novas camadas. Porém, a tendência é perder performance em prol da generalidade. Dessa forma, foi decido desenvolver a solução inicialmente em uma única plataforma.

Buscandos-se microcontroladores disponíveis à baixo custo com potencial desempenho esperado encontrou-se as linhas ARM Cortex-M e ESP32. Comparando-se custos, a série ESP32 ganha com grande vantagem ao disponibilizar módulos com processador dual-core, WiFi, BLE, e 8MB de RAM externa por cerca de US$ 10,00 no mercado de varejo brasileiro. Não pode-se encontrar nenhum similar em quantidade de características de conectividade e processamento pelo custo citado.

Por conta do ESP32 ser desenvolvido pela empresa Espressif, o framework escolhido para o desenvolvimento foi o framework oficial esp-idf, na versão v5.0 enquanto este trabalho é desenvolvido. A suficiente documentação (ESPRESSIF, 2023a) e larga quantidade de módulos, bem como o código aberto (ESPRESSIF, 2023c) do framework, que permite o debug de falhas inesperadas, reforçam a decisão.

3.2 Escolha do classificador de estudo

O modelo Binary de (FERNANDES; CORDEIRO; RECAMONDE-MENDOZA, 2021), publicado há um ano da data de início desse trabalho consiste em um modelo de classificador com rede neural convolucional aplicada a uma matriz proveniente de um espectrograma em escala mel.

Por conta do software do trabalho mencionado ter sido publicado como código aberto, como pelo fato de usar conjunto de dados aberto (MUKUNDARAJAN *et al.*, 2017), foi eleito para desenvolvimento do classificador embarcado que é objeto deste trabalho.

Essas características permitem comparar os resultados de inferência implementados em um cenário controlado, utilizando um modelo efetivamente usado para o propósito de detecção de Aedes aegypti. O número de parâmetros no modelo inteligente é de 1.137.858 em float32, ocupando em torno de 4,34 MB.

3.3 Engenharia reversa da implementação do classificador binary

O principio se deu com a análise do código publicado em repositório no GitHub (FERNANDES; CORDEIRO, 2020). Preliminarmente pode-se observar o uso da API Keras do TensorFlow 2, bem como o uso das bibliotecas numpy e librosa para extração de características.

A entrada da rede neural é uma matriz com valores normalizados entre 0 a 1 do espectrograma em escala mel que corresponde a um array de formato (bandas, quadros, 1) onde bandas é 60 e quadros é 40. Bandas representa o número de bandas na escala mel, enquanto quadros representa o número de subsegmentos processados para produção do espectrograma. Outros parâmetros são passados ao método librosa.feature.melspectrogram(...), como o período da FFT e comprimento de salto, que são 1024 e 256, respectivamente.

3.3.1 Parâmetros ocultos para extração de características

Entretanto, esses parâmetros não são suficientes para produzir o espectrograma. Averiguando as entranhas da librosa, observam-se vários parâmetros padrões nas chamadas internas das funções em parâmetros opcionais. Isso é inicialmente complicado de se perceber pois o parâmetro genérico **kwargs permite passar como dicionário parâmetros de forma pouco explicita ao engenheiro desabituado com a base de código, ou são parâmetros distribuídos dentro de várias chamadas. Estudando, encontram-se outros 3 fatores que estão implícitos na biblioteca para reproduzir a implementação conforme subseções 3.3.1.1, 3.3.1.2 e 3.3.1.3.

3.3.1.1 Potência ou módulo

O primeiro deles se dá com a potência no qual os valores do espectrograma são elevados. O padrão para espectrograma é usar o expoente 2 na magnitude dos elementos de resultado da FFT, a fim de obter-se a potência do sinal. Enquanto isso, a função subjacente usada para calcular-se o espectrograma é a rfft que entrega um array de módulos dos elementos da FFT. Portanto, a implementação do algoritmo deve utilizar o quadrado dos módulos dos coeficientes obtidos pela FFT.

3.3.1.2 Preenchimento

O segundo é o preenchimento (ou *padding*). O preenchimento está melhor detalhado na seção 2.7. Essa manipulação pode ser percebida antes mesmo da chamada da função librosa.feature.melspectrogram(...), no cálculo da quantidade de amostras passadas de cada segmento de áudio no qual se fará a extração de características.

Caso houvesse ausência de preenchimento, o número de quadros seria exatamente $(n\_amostras - NFFT)/hop\_length + 1$. Por sua vez, o $n\_amostras$ deveria ser $(n\_frames - 1) \cdot hop\_length + NFFT$. Todavia, observa-se no código analisado que o $n\_amostras$ foi considerado como $= (n\_frames - 1) \cdot hop\_length$, descontando o período da FFT.

Isso ocorre graças ao preenchimento, que concatena NFFT/2 em ambas as extremidades do segmento antes de prosseguir com a STFT. Dessa forma, a metade da primeira e a última sequência passada para a FFT consistem de valores preenchidos. Existem diferentes formas de se definir os valores acrescentados por preenchimento. O preenchimento padrão da librosa versão 0.8.1, e por ausência de parâmetro específico no código analisado, a utilizada nesse trabalho, é a reflexão.

Nota: A reflexão ser o método padrão de padding só é verdade para a versões <0.9.0 da librosa, como a 0.8.1, que é usada no ambiente de desenvolvimento deste trabalho. É certo que durante o desenvolvimento da rede neural de (FERNANDES; CORDEIRO, 2020) também o parâmetro padrão era reflexão, pois esse fora alterado para constante (*zero padding*) apenas na versão 0.9.0 da librosa, conforme pode-se observar no pull request #1362 no GitHub oficial da biblioteca (MCFEE, 2021). Sendo essa mudança de março de 2021, é posterior ao desenvolvimento do classificador original.

A reflexão se dá espelhando as extremidades, copiando NFFT/2 dos primeiros elementos, invertendo-se, e então concatenando ao início do vetor que será passado à STFT. O mesmo é feito para os NFFT/2 últimos elementos, porém concatenando ao final do vetor.

3.3.1.3 Janela

O terceiro parâmetro que está oculto é o tipo de janelamento, ou windowing, usada na FFT. Novamente, a presença do parâmetro interno padrão é difícil de se encontrar e não está plenamente documentado pela librosa.feature.melspectrogram(...), porém pode-se descobrir que é a janela de Hann (KAHLIG, 1993).

Tabela 2 – Parâmetros para Espectrograma em escala mel

Parâmetros	**Valores**
bandas	60
quadros	40
comprimento do salto	256
número de amostras por segmento	$(hop\_length) \cdot (n\_frames - 1) = 9984$
período da FFT	1024
taxa de amostragem	8000 Hz
potência	2.0 (padrão)
preenchimento	reflexão (padrão na librosa 0.8)
janela de análise	hann (padrão)

Fonte: o Autor, 2023.

3.4 Alteração do script para exportação do modelo

O script binary.py (FERNANDES; CORDEIRO, 2020) foi alterado a fim de exportar o modelo keras como arquivo json e os pesos em formato HDF5 usando os métodos to_json(...) e save_weights(...) da API Keras. O script por padrão usa StratifiedKFold, treinando 10 vezes em subconjuntos de treinamento e testes estratificados. A fim de conduzir esse trabalho os parâmetros gerados pelo primeiro *fold* foram considerados.

3.5 Análise do Hardware comercialmente disponível

No mercado encontram-se inúmeros microcontroladores, porém não foi possível encontrar módulo com custo similar que disponha de transceptor WiFi, memória flash de 4MB e memória RAM total de 8MB, endereçada diretamente até 4MB. A quantidade de memória flash e RAM disponível não são imediatamente as mais adequadas, mas ainda permitem margem para execução da rede disponível através de otimizações. Além disso o datasheet do ESP32 informa que esse tem unidade de ponto flutuante (FPU) e é dual-core, o que permite executar a classificação em uma thread diferente da gravação, não ocorrendo bloqueamento. Portanto, o uso do módulo ESP32 WROVER, que além do SoC ESP32, possuí CI de memória flash e memória DRAM conectada por barramento SPI foi definido como requisito deste projeto.

3.6 Busca por métodos de conversão de rede em API Keras para código C

É conhecido que os microcontroladores em que se pretende implementar o sistema classificador possuem restrição na quantidade de memória RAM disponível bem como de clock da CPU. Naturalmente, pelo Keras ter sido integrado ao TensorFlow 2 como API

de alto nível, um primeiro candidato seria o TensorFlow Lite. Entretanto, o TensorFlow Lite é uma biblioteca bastante completa, com inúmeras camadas de aplicação, tornando-a humanamente mais custosa para alterações.

Em relação ao tamanho do binário compilado da biblioteca, houve certa preocupação inicial, com a biblioteca compilada para alguns modelos comuns ocupando em torno de 300KB (GOOGLE, 2022a). Isso poderia ser melhorado com a versão do TensorFlow Lite para microcontroladores que pode ter um runtime de apenas 16KB para um processador ARM.

Entretanto os parâmetros da primeira camada dense do modelo de interesse tem tamanho superior à memória mapeada com acesso direto do ESP32 WROVER, dispositivo escolhido pela disponibilidade de unidades para início do trabalho. Isso inviabiliza sua implementação sem alteração do modelo no TensorFlow Lite.

O modelo poderia ser alterado utilizando-se de quantização para reduzir o espaço ocupado, porém havendo delimitação do escopo, não se pretende nesse momento modificar a rede neural artificial. Esse trabalho se atem a implementar um algoritmo equivalente ao que fora previamente desenvolvido em python.

Após pesquisa inicial encontrou-se a Keras2C(CONLIN, 2022), uma biblioteca simples, que implementa em funções atômicas, camadas usadas no modelo como Conv2D, MaxPool, Flatten e Dense1D. Também fornece um script gerador de código C para modelo Keras em formato json. Passando alguns testes de sanidade, essa foi eleita para a implementação. Apresenta grande versatilidade e simplicidade na modificação após geração do código, sendo adequada ao objetivo de adaptar o software aos recursos de hardware disponíveis.

3.7 Conversão usando Keras2C

O modelo foi convertido utilizando-se a biblioteca Keras2c disponível à época (commit Sha1: 653745a1634dbea9c42a88505073008c9482585e), de 14/Jul/2021 (CONLIN, 2022). Entretanto ao iniciar seu uso percebeu que algumas funcionalidades teriam de ser adicionadas. A exportação do modelo oferecia duas opções:

1) escrever os parâmetros em grandes vetores estáticos que aumentariam o tamanho da memória de programa necessária, impossibilitando a gravação no ESP32 WROVER que não possui memória de programa suficiente;

2) escrever os parâmetros em arquivos CSV que são lidos em runtime e carregados em vetores dinamicamente alocados.

3.7.1 Parâmetros carregados dinamicamente

Descartada a primeira opção, mesmo a segunda se apresenta limitada. Isso porque realiza conversão dos pesos, que são float32, para uma string decimal correspondente, justamente pela natureza textual do formato CSV. Nesse processo pode haver truncamento e perdas na conversão para string decimal e depois novamente do caminho reverso para float32 IEEE 754. Definiu-se que os parâmetros da rede neural deveriam ser armazenados em arquivos binários para evitar perdas, em sequencia little-endian, a mesma endianess dos processadores Tensillica Xtensa LX6, quanto dos processadores x86_64 dos computadores utilizados para gerar os arquivos.

3.7.2 Inferência com carregamento sob demanda de parâmetros

Como a memória flash dos módulos disponíveis comercialmente é de apenas 4 MB, há então necessidade de fazer-se o carregamento dinâmico dos parâmetros da rede neural a partir de bloco externo.

O ESP32 apresenta três drivers SPI de hardware, o que permite fácil comunicação com um cartão SD com resistores de pull-up conforme especificações, além do framework esp-idf já apresentar bibliotecas prontas para mapeamento do cartão SD com sistema de arquivos FAT32.

Conforme relatado, o ESP32 original não é capaz de mapear mais de 4MB de RAM externa, apesar do módulo ESP32 WROVER oferecer um circuito integrado de 8 MB de RAM externa. Dadas essas premissas há algumas opções:

3.7.2.1 Carregamento parcial por leitura sequencial do cartão SD

O código fonte gerado pelo utilitário do Keras2C foi então modificado de forma a fazer o carregamento dos parâmetros sob demanda em tempo de inferência, alocando e desalocando memória conforme necessário. Todavia, a primeira camada completamente conectada ("dense") possui 1.081.600 parâmetros em float32 que ocupam sozinhos cerca de 4,13 MB, ultrapassando a quantidade total de memória externa diretamente mapeável.

O tensor de entrada desta camada é unidimensional e seu kernel bidimensional, apresentando 4224 linhas e 256 colunas, além de ter o vetor de bias de 256 parâmetros.

Seja X o vetor unidimensional da entrada, B o kernel e Y o dot(X,K) + B, conforme definido pela documentação do TensorFlow, as equações 35, 36, 37 . Para simplificação, a função de ativação está omitida, porém, é calculada após as somas ponderadas no algoritmo:

$$\mathbf{Y_{(1,o)}} = \mathbf{X_{(1,n)}} \cdot \mathbf{K_{(n,o)}} + \mathbf{B_{(1,o)}} \tag{35}$$

$$y_{1,j} = \sum_{i=1..n} x_{1,i} \cdot k_{i,j} + b_{1,j} \tag{36}$$

$$\begin{bmatrix} \boldsymbol{y_{1,1}} \\ y_{1,2} \\ ... \\ y_{1,o} \end{bmatrix}^T = \begin{bmatrix} \boldsymbol{x_{1,1}} \\ \boldsymbol{x_{1,2}} \\ \boldsymbol{...} \\ \boldsymbol{x_{1,n}} \end{bmatrix}^T \cdot \begin{bmatrix} \boldsymbol{k_{1,1}} & ... & k_{1,o} \\ \boldsymbol{k_{2,1}} & ... & ... \\ \boldsymbol{...} & ... & ... \\ \boldsymbol{k_{n,1}} & ... & ... \end{bmatrix} + \begin{bmatrix} \boldsymbol{b_{1,1}} \\ b_{1,2} \\ ... \\ b_{1,o} \end{bmatrix}^T \tag{37}$$

Esse cálculo seria implementado pelo algoritmo 1 de acordo com o seguinte pseudocódigo:

Algoritmo 1: Camada dense

Entrada: $X[n], K[n][o], B[o], n, o$
Saída: $Y[o] = dot(X, K) + B[o]$
início
  $memset(\&Y, 0, tamanho(Y))$;
  $i \leftarrow j \leftarrow 0$;
  repita
    repita
      $Y[j] \leftarrow Y[j] + X[i] \cdot K[i][j]$;
      $j \leftarrow j + 1$;
    até $j = o$;
    $i \leftarrow i + 1$;
  até $i = n$;
  $j \leftarrow 0$;
  repita
    $Y[j] \leftarrow Y[j] + B[j]$;
    $Y[j] \leftarrow ativação(Y[j])$;
    $j \leftarrow j + 1$;
  até $j = o$;
fim

A forma encontrada para reduzir a necessidade de necessidade de alocar todo a matriz K na memória RAM, sem depender de leitura randômica do cartão SD, foi dividir o processamento em parcelas de acordo com o vetor de entrada.

O algoritmo foi alterado, com a adição do parâmetro inteiro Q, representando o número de linhas da matriz K a ser carregado por por vez. Assim a memória necessária para guardar o K na memória é apenas $Q \cdot o \cdot tamanho(float32)$. O pseudocódigo ilustra o raciocínio do algoritmo 2:

Algoritmo 2: Camada dense carregando parâmetros parcialmente

Entrada: $X[n], K_{sdcard}[n][o], B[o], n, o, Q$
Saída: $Y[o] = dot(X, K) + B[o]$
início
  $memset(\&Y, 0, tamanho(Y))$;
  $i \leftarrow j \leftarrow 0$;
  repita
    //carrega linha da matriz K_{sdcard} para a memória RAM;
    se *(resto i por Q) == 0* **então**
      $K\_linha\_inicio \leftarrow i$;
      $K \leftarrow K_{sdcard}[i : max(i + Q, n)]$ //dimensionalidade (Q,o) ;
    repita
      $Y[j] \leftarrow Y[j] + X[i] \cdot K[i - K\_linha\_inicio][j]$;
      $j \leftarrow j + 1$;
    até $j = o$;
    $i \leftarrow i + 1$;
  até $i = n$;
  $j \leftarrow 0$;
  repita
    $Y[j] \leftarrow Y[j] + B[j]$;
    $Y[j] \leftarrow ativação(Y[j])$;
    $j \leftarrow j + 1$;
  até $j = o$;
fim

3.7.2.2 Armazenamento de parâmetros na zona de memória RAM indiretamente acessível do ESP32

Apesar do ESP32 não permitir o mapeamento direto de todos os 8MB do módulo de RAM SPI, o framework de desenvolvimento da Espressif, no componente esp-psram (ESPRESSIF, 2023e), fornece uma API para chaveamento de bancos de memória (bank switching). A implementação dela necessita que o mapeamento e alocação sejam feitos em intervalos de 32 KB, por uma simplificação da implementação usar o cache inteiro da unidade de gerenciamento de memória (MMU).

O uso da funcionalidade se dá selecionando a definição de compilação CONFIG_SPIRAM_BANKSWITCH_RESERVE para um valor inteiro entre 1 e 62 e chamando alguma das funções da API, fazendo o *linker* ligar a API.

O número de blocos reservados para bank switch reduz o espaço disponível para o alocador ordinário do heap, mas deixa disponível para acesso através da API himem.

É importante observar que há uma falha catastrófica na implementação do alocador padrão da RAM externa que impossibilita o funcionamento conjunto com a API himem até pelo menos a versão v5.0 do esp-idf. Os endereços são conflitantes e conduzem a violações de acesso. Após inspeção minuciosa e debug da biblioteca, pode-se corrigir a falha, que consistia em limites de endereçamento do heap mal calculados ao ter-se a API himem usada.

Coincidentemente, o *kernel* da camada dense tem um tamanho divisível por 32 kB, totalizando exatamente 132 blocos. É evidente que os 4 MB correspondem a apenas 128 blocos reservados, assim fazendo necessário a reserva de ao menos 4 blocos a fim expandir a memória disponível à API. Sabe-se que, pelo tamanho de cada linha do *kernel* ser 256 números de tipo float32 e ocupar 1 kB, cada bloco contém um número inteiro de linhas do *kernel*, que ao todo são 4224.

Assim, um ensaio foi conduzido dividindo-se o acesso em 33 partes com 4 blocos cada, equivalente ao processamento de 128 linhas por mapeamento. Também, a fim de explorar o possível ganho de desempenho com a redução no número de chamadas das rotinas de mapeamento de desmapeamento se testou com um menor número de partes, como por exemplo 12 partes com 11 blocos cada, e 4 partes com 33 blocos cada. Lembra-se aqui da desvantagem: para mapear-se mais blocos, mais memória deverá ser reservada para a API himem, ficando indisponível ao alocador padrão que tem alinhamento de 8 bits, em oposição aos 32 kB que a API obriga.

Usando a API himem, então, se desfaz a necessidade de acesso ao cartão SD após os parâmetros estarem todos carregados na RAM externa, mesmo que em regiões acessíveis apenas por bank switching.

O algoritmo 3 demonstra como esse procedimento é realizado:

Algoritmo 3: Camada dense usando API himem

Entrada: $X[n], K_{sdcard}[n][o], B[o], n, o, P, nBLKs$

Saída: $Y[o] = dot(X, K) + B[o]$

início

$memset(\&Y, 0, tamanho(Y))$;

$i \leftarrow j \leftarrow 0$;

repita

//carrega linha da matriz K_{sdcard} para a memória RAM ;

se *(resto i por (32 · P)) == 0* **então**

$K\_linha\_inicio \leftarrow i$;

$nBlockIndice \leftarrow i/(32 \cdot p)$;

$himem\_map(nBlockIndice, nBLKs, \&K)$ //map nBLKs blocos;

repita

$Y[j] \leftarrow Y[j] + X[i] \cdot K[i - K\_linha\_inicio][j]$;

$j \leftarrow j + 1$;

até $j = o$;

$i \leftarrow i + 1$;

até $i = n$;

$j \leftarrow 0$;

repita

$Y[j] \leftarrow Y[j] + B[j]$;

$Y[j] \leftarrow ativação(Y[j])$;

$j \leftarrow j + 1$;

até $j = o$;

fim

4 Metodologia Experimental

4.1 Ambiente de Desenvolvimento

Para os ensaios deste trabalho as versões dos componentes de software utilizado estão dispostos na tabela 3.

Tabela 3 – Versões de componentes utilizados no ambiente

Componente	**Versão**
python	3.9.1
numpy	1.19.5
tensorflow	2.5.0
librosa	0.8.1
esp-idf	5.0*
keras2c	14/Jul/2021*
esp32-fft	12/Dez/2017*

* Versões de referência, componentes foram modificados conforme descrito no decorrer deste trabalho.

O hardware escolhido fora um módulo ESP32 WROVER conectado a um cartão por barramento SPI com resistores de *pull-up* de 10 kΩ, conforme determina o *datasheet* do ESP32.

4.2 Hipóteses

A partir das análises realizadas as seguintes perguntas foram encontradas:

1. A implementação desenvolvida pode realizar as mesmas operações sem encontrar instabilidade numérica comprometendo os resultados?

2. Considerando que o ESP32 não possui aceleração de hardware para precisão dupla (float64 ou double) e apresenta uma unidade de ponto flutuante com algumas operações para precisão simples (float32), qual o ganho de performance ao executar a FFT usando precisão simples em vez de precisão dupla?

3. É necessário usar precisão dupla para os tipos das variáveis usadas dentro da FFT? Sabe-se que a biblioteca librosa internamente utiliza precisão simples para calcular a conversão de espectrograma para escala mel, enquanto que a FFT utiliza a precisão padrão da linguagem python, a precisão dupla.

4. Qual o tempo de execução do classificador desde a extração de características até o resultado do processamento?

5. Qual o impacto nos resultados com a compilação em otimização para tamanho (-Os) e otimização para velocidade (-O2) no compilador xtensa-esp32-elf-gcc disponibilizado com o framework esp-idf?

4.3 Ensaios

4.3.1 Analisar erro numérico de 240 segmentos de áudio a partir da base de dados original

1. Preparar um subconjunto de 240 segmentos de 9984 amostras de áudio, amostrados a 8 kHz, portanto, cada um com 1,248 segundos. A metade deles correspondentes à classe negativa (NÃO É AEDES AEGYPTI) e a outra metade à classe positiva (É AEDES AEGYPTI). Sendo 120 segmentos da classe positiva e 120 segmentos da classe negativa, selecionados usando a biblioteca numpy e uma semente de aleatoriedade escrita a esmo.

2. Exportar os 240 segmentos para um arquivo binário salvo no cartão SD e conectá-lo ao MCU.

3. Programar rotina para executar esse ensaio específico. Os requisitos são: i) fazer o carregamento individual dos segmentos; ii) executar o classificador e iii) salvar a saída númerica (antes de definir a saída categórica, pelo elemento de maior valor) deste no mesmo cartão SD para cada segmento de áudio disponível no cartão SD.

4. Definir a configuração da FFT para usar vetores em precisão dupla, como faz o algoritmo da biblioteca de referência, a librosa.

5. Compilar, gravar o novo firmware, obter os resultados e carrega-los em um Jupyter Lab notebook onde podem ser comparados com as saídas do classificador pré-existente em python.

6. Obter métrica de similaridade dos resultados: Obter média, mediana e desvio padrão da diferença entre a saída do classificador ESP32 e a saída do classificador python. Comparar se as classes obtidas são similares.

$$\mathbf{Erro_{(240,2)}} := \mathbf{Saída_{Python(240,2)}} - \mathbf{Saída_{ESP32(240,2)}} \tag{38}$$

A saída da rede neural *Binary* de (FERNANDES; CORDEIRO; RECAMONDE-MENDOZA, 2021) tem uma camada de saída com 2 valores probabilísticos, cada qual corresponde a uma das classes: (i) NÃO É AEDES e (ii) É AEDES. A classe é definida como a correspondente ao valor maior. Para fins de comparação dos resultados do ensaio, o erro é definido em termos das saídas numéricas da rede neural.

A quantia de 240 segmentos foi definida ao se limitar à duração de cada ensaio em 2 horas. A partir de testes iniciais, o limite do tempo de classificação total, i.e. tempo extração de características e inferência pela rede neural convolucional, foi estimado em 30 segundos. Assim, 240 execuções do algoritmo seriam possíveis. De forma a equilibrar o peso dos segmentos correspondentes à amostras positivas e negativas, os segmentos coletados foram estratificados pelo resultado desejado.

4.3.2 Avaliar diferença de tempo de execução com extração de características usando precisão simples (float32) e precisão dupla (float64 ou double)

1. Criar rotina que carrega uma amostra de áudio, chama o classificador e imprime na saída serial do MCU o tempo de execução.

2. Executar o firmware compilado usando precisão simples e então com precisão dupla por N vezes.

3. Anotar em cada caso do passo anterior as o tempo de execução a partir do tempo registrados na saída serial.

Após carregamento do FreeRTOS, sistema operacional base do esp-idf, o tempo exibido na saída serial é contabilizado pelos ticks do FreeRTOS, que por padrão tem período de 10ms, sendo então a resolução temporal esperada.

O número N de execuções repetidas do firmware em cada caso foi definido em 3 para estimar a variabilidade, embora seja esperado um tempo de execução bastante uniforme pela ausência de outras threads interferindo na execução do algoritmo.

4.3.3 Obter saídas usando precisão simples para cálculo da FFT para os dados de entrada no Ensaio 1

1. Repetir procedimento de classificação para os segmentos de áudio selecionados no Ensaio 1, porém com o firmware compilado para executar a FFT em precisão dupla.

2. Gerar medidas descritivas das diferenças entre as saídas de cada caso com a saída gerada pelo classificador em python.

3. Comparar as medidas do erro (definido como a diferença entre a saída numérica produzida pelo classificador embarcado e a saída produzida pelo classificador em python) com as medidas da diferença interna (definida como a diferença entre a saída numérica do primeiro e do segundo neurônio da camada de saída).

4.3.4 Análise do tempo de execução com a mudança de parâmetros

1. Repetir o Ensaio 2 variando a precisão da FFT (simples e dupla), o parâmetro de otimização do compilador (¨Osë ¨O2), e o número de blocos reservados e mapeados usando a API himem (4, 12 e 33).

2. Executar N vezes e anotar o tempo de execução para cada caso.

Novamente o N escolhido foi 3 de forma a estimar a variabilidade no tempo de execução, se houver, que espera-se ser baixa.

5 Resultados

5.1 Erro de inferência

A partir do subconjunto dos 240 segmentos de dados a classe prevista (sendo definida pelo elemento do vetor de saída de maior magnitude), em ambos os cenários, usando FFT com precisão simples e precisão dupla, foi exatamente igual ao previsto no algoritmo python pré-existente. As métricas descritivas do erro dos valores da saída da rede neural para características extraídas com FFT em precisão dupla são retratadas na tabela 4.

Tabela 4 – Métricas descritivas do erro com precisão dupla na FFT

Métrica	**Erro**
Média	$2,8 \cdot 10^{-10}$
Mediana	0
Desvio Padrão	$1,5 \cdot 10^{-7}$
Mínimo	$-1,0 \cdot 10^{-6}$
Máximo	$1,1 \cdot 10^{-6}$

Fonte: o Autor, 2023.

A média do erro é menos de 2% do desvio padrão, mostrando ausência de inserção de viés. Considerando que as saídas da rede neural usam função de ativação sigmoide, cuja imagem é [0 ; 1]. Também que números de ponto flutuante IEEE 754 de precisão simples possuem 23 bits na mantissa. A média de saídas do conjunto de dados é 0,53. Em IEEE 754 32 bits, a representação se dá com expoente -1 e 23 bits de mantissa: $1, b_2 2 b_2 1 b_2 0 ... b_0 \cdot 2^{-1}$. A resolução em torno de 0,53 é portanto $1 \cdot 2^{-24}$ que equivale a aproximadamente $6 \cdot 10^{-8}$. Dessa forma, o desvio padrão é inferior a 3 vezes a resolução absoluta da média do sinal, tornando plausível desconsiderar o erro.

5.2 Tempo de Extração de Características

Na tabela 5 estão os resultados obtidos ao medir-se o tempo total de construção da STFT a partir da FFT de precisão simples e dupla, conversão para Espectrograma Mel, e normalização conforme definida anteriormente.

A resolução da função utilizada para medir o tempo de execução (uint32_t esp_log_timestamp()) usa ticks do FreeRTOS. Por padrão, no framework esp-idf a duração de cada tick é de 10 ms.

Assim, a incerteza padrão de tipo A atribuída foi de 3 ms ao considerar o desvio padrão da distribuição retangular ($\frac{10}{\sqrt{12}}$ ms). A incerteza padrão de tipo A é a exibida em todos os resultados os quais não foi detectada variabilidade no tempo de execução (isto é, devio padrão amostral nulo). A incerteza padrão de tipo B é a que está acompanhada do resultado sempre for superior à incerteza de tipo A.

Tabela 5 – Tempo de execução em segundos do espectrograma mel com diferentes precisões na FFT em milissegundos acompanhado da incerteza padrão

Precisão	Otimização no compilador	
	-Os	**-O2**
simples	723 ± 6	700 ± 3
dupla	2360 ± 3	2290 ± 3

Fonte: o Autor, 2023.

É importante notar que, nesse ensaio, estavam sendo utilizados os parâmetros padrões do framework para frequência de relógio do barramento SPI da memória externa, frequência de relógio da CPU, e correções de software para falhas em versões mais antigas da MMU. O tempo é melhor explorado na seção 5.4 quando esses parâmetros foram alterados.

Como se pode observar, o tempo mudando-se o tipo das variáveis usadas na FFT para precisão simples, como está no código original da esp32-fft, implicou numa redução de 69% no tempo de execução em relação à variante em precisão dupla. A documentação do SoC informa que há uma FPU disponível no ESP32, porém na documentação (ESPRESSIF, 2023d) do FreeRTOS modificado pela Espressif é informado que as operações com precisão dupla são feitas por software, sem o uso da FPU, o que explica a discrepância entre os tempos observados.

5.3 Erro de Inferência com precisão simples

A partir da grande redução no tempo de extração de características no ensaio 2, decidiu-se analisar o quanto essa redução na precisão poderia impactar o erro na saída do classificador. Na tabela 6 estão as métricas descritivas do erro obtido com FFT calculada usando precisão simples. Na tabela 7 encontram-se os resultados com ambos os erros, o que permite observar uma redução de 75% no desvio padrão do erro, que se mantém com a média em 0. No histograma da figura 12 pode-se ter uma comparação visual.

5.4 Tempo de execução

No ensaio do tempo de execução variando parâmetros pode-se observar alguns parâmetros ativados por padrão no esp-idf com alvo no esp32 que podem ser modificados:

Tabela 6 – Métricas descritivas do erro com precisão simples na FFT

Métrica	**Erro**
Média	$-4,7 \cdot 10^{-9}$
Mediana	0
Desvio Padrão	$6,2 \cdot 10^{-7}$
Mínimo	$-3,6 \cdot 10^{-6}$
Máximo	$3,5 \cdot 10^{-6}$

Fonte: o Autor, 2023.

Tabela 7 – Métricas descritivas do erro com precisão simples e precisão dupla na FFT

Métrica	**Precisão Simples**	**Precisão Dupla**	|razão|
Média	$-4,7 \cdot 10^{-9}$	$2,8 \cdot 10^{-10}$	6%
Mediana	0	0	-
Desvio Padrão	$6,2 \cdot 10^{-7}$	$1,5 \cdot 10^{-7}$	25%
Mínimo	$-3,6 \cdot 10^{-6}$	$-1,0 \cdot 10^{-6}$	28%
Máximo	$3,5 \cdot 10^{-6}$	$1,1 \cdot 10^{-6}$	30%

Fonte: o Autor, 2023.

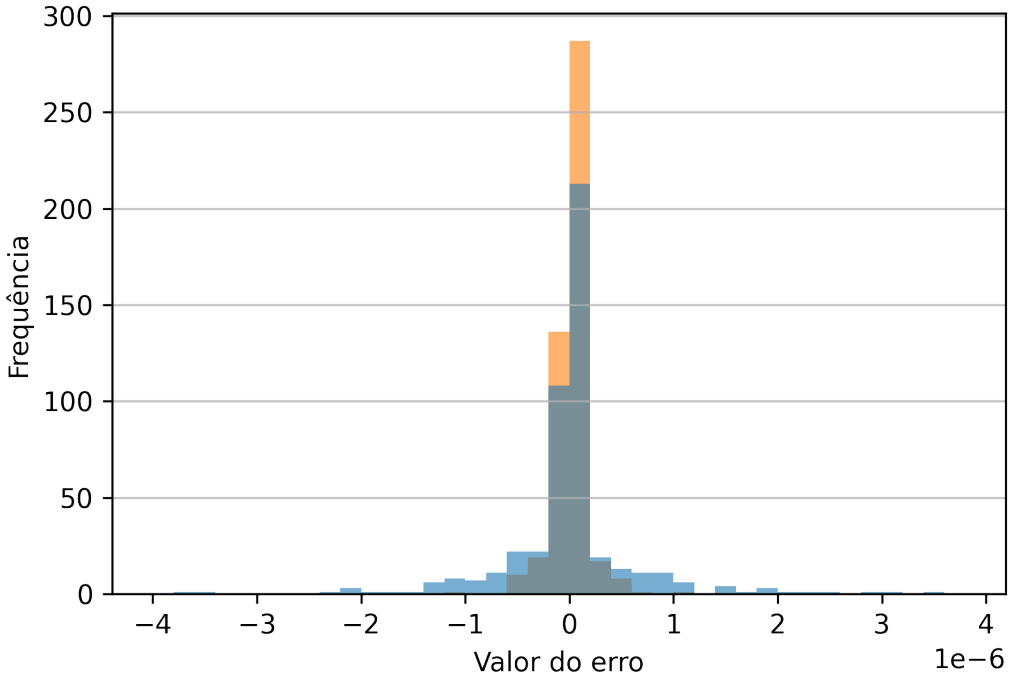
(i) A frequência de relógio da CPU, que por padrão é 160 MHz, pode ser aumentada para 240 MHz; (ii) o *workaround* de software feito para evitar problemas no cache da RAM externa, que fora corrigido na terceira revisão do esp32; (iii) a frequência do barramento SPI da RAM externa, que por padrão é 40 MHz, mas pode ser aumentado para 80 MHz desde que a frequência da memória flash também seja.

Esses parâmetros não são compatíveis com todas as versões do esp32, porém o alvo de compilação é apenas um, por conta disso, para manter compatibilidade, o framework original coloca por padrão a configuração compatível com todos os dispositivos programados pelo mesmo alvo.

O dispositivo adquirido para esse ensaio é da revisão v3.0, o que não apresenta a falha de cache (ESPRESSIF, 2023b). Também não é um ESP32-S0WD que tem limitação de frequência da CPU, e por testes preliminares pode-se confirmar a possibilidade de aumentar a frequência da RAM e por conseguinte da memória flash sem instabilidade. Dessa forma, além dos parâmetros planejados, a frequência da RAM, a frequência da CPU e a ativação da flag de workaround do cache da memória RAM foram alterados.

O tempo de execução iterando diversos parâmetros está descrito na tabela 8 no apêndice A. A partir desses dados, todos eles com incerteza padrão de 3 ms por conta da resolução de 10 ms da função usada para medição do tempo, pode-se observar alguns comportamentos.

Figura 12 – Histograma do Erro do Classificador



Em laranja está o histograma do resultado do erro das amostras usando FFT em precisão dupla, enquanto que em azul o histograma usando FFT em precisão simples. Em cinza está a região em que ambos os histogramas se sobrepõem.
Fonte: o Autor, 2023.

5.4.1 Figuras

Os ensaios estão codificados nas figuras 13, 14 e 15 em uma legenda cujos termos separados por hífen significam respectivamente: (i) Frequência do barramento SPI da RAM em MHz; (ii) correção via software da falha no cache (memw é um método, no é ausência de correção por software); (iii) frequência da CPU do esp32 e (iv) otimização de compilação (-Os indica tamanho e -O2 indica performance).

5.4.2 Tempo de Extração de Características

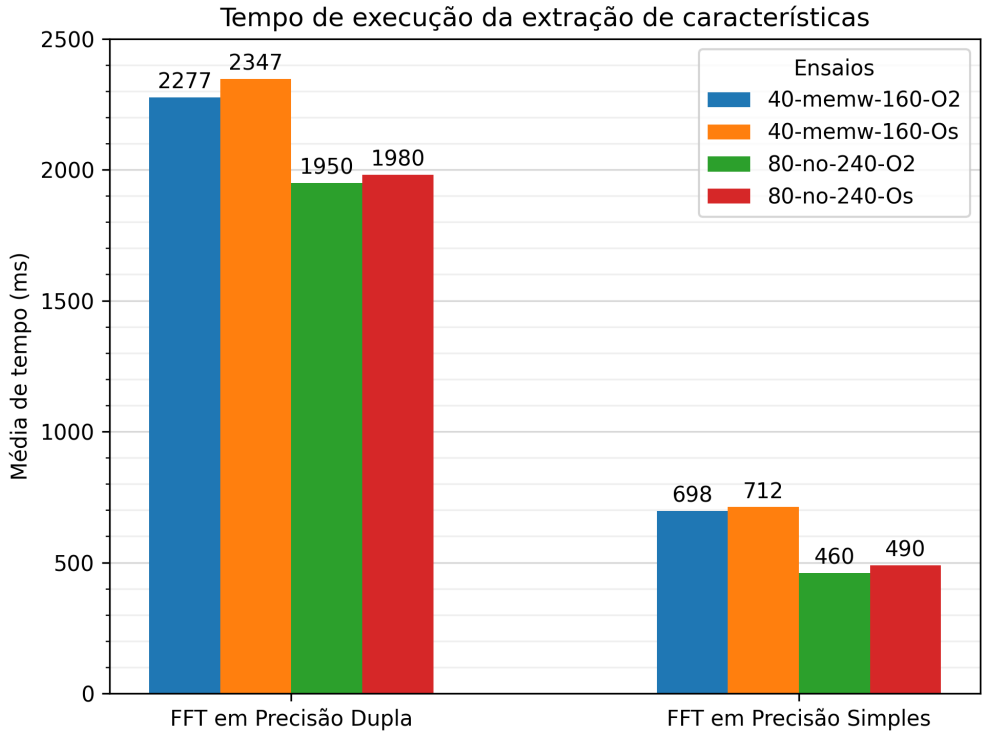
A figura 13 resume os achados. Dentre eles podemos comentar que o tempo de extração de características foi pesadamente afetado pela precisão da FFT, seguindo o que fora previsto em ensaio anterior. O tempo foi reduzido em torno de 69% a 75%.

Apesar do erro ligeiramente maior da FFT comparado ao resultado produzido pela rotina original em python (que usa precisão dupla por padrão), recomenda-se portanto projetos usem a precisão simples visto que são aceleradas por unidade específica no ESP32, enquanto que operações em precisão dupla podem ser apenas realizadas por uma abstração de software que toma muito mais ciclos de relógio.

Quanto à compilação em otimização para performance contra otimização para tamanho, pouco pode-se observar em ganho de desempenho na geração do espectrograma mel, sempre estando abaixo de 5%.

Em relação ao conjunto de parâmetros de hardware padrão, frequência da RAM, correção de cache e freq. da CPU, encontrou-se uma redução imediata do tempo de execução entre 31% a 35%.

Figura 13 – Tempo de execução da FFT em alguns dos ensaios realizados



Legenda conforme seção 5.4.1.
Fonte: o Autor, 2023.

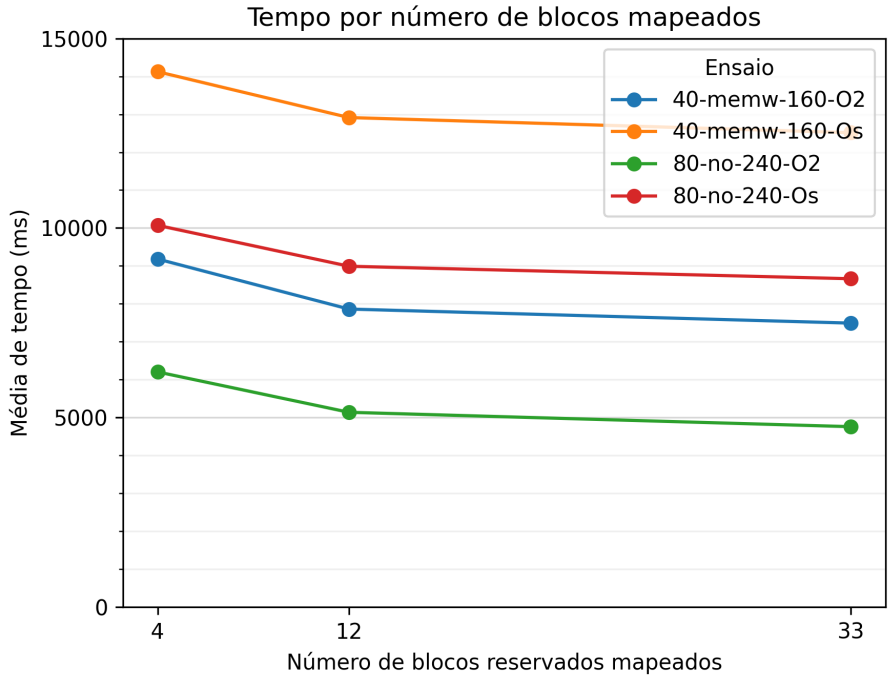
5.4.3 Tempo de Inferência da Rede Neural Binary

Confirmou-se a hipótese do acesso usando a API himem apresentar overhead no acesso à regiões mapeadas. Como se pode ver na figura 14, aumentado-se o número de blocos acessados por vez percebe-se redução no tempo de acesso. A saber, no esp-idf v5.0 a API himem usa blocos de 32 kB, sendo o kernel da camada "dense"ocupa ao todo 132 blocos. Por conta disso o número de chamadas às funções de mapeamento e desmapeamento são inversamente proporcionais ao número de blocos reservados para a API. Quando 4 blocos são reservados são feitas $^{132}/_{4} = 33$ iterações, sendo que com 12 blocos $^{132}/_{12} = 11$ iterações e por fim com 33 blocos $^{132}/_{33} = 4$ iterações.

O uso de apenas 4 blocos prevê a ocupação total e plena do espaço reservado pela API himem, porém, não produz o melhor resultado em termos de tempo de processamento. Na figura 15 tem-se as curvas dos mesmos ensaios. Pode-se encontrar linearidade entre o

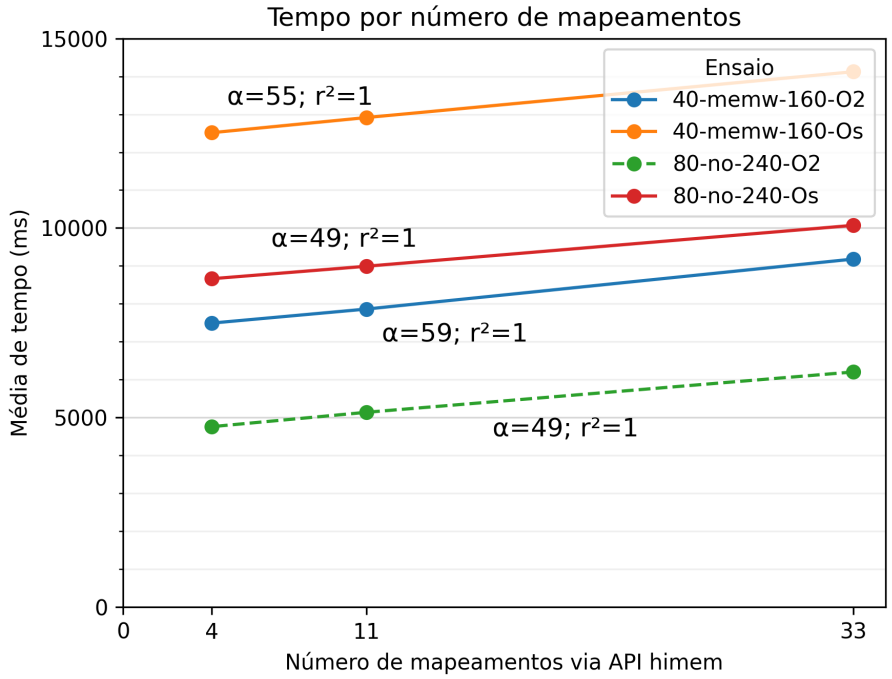
número de ciclos de mapeamento e desmapeamento da API himem ao dispo-los no eixo horizontal. A regressão linear das curvas plotadas indicam que o tempo adicional é de 49 ±3 ms a 59 ±3 ms por ciclo de remapeamento.

Figura 14 – Tempo de execução da Rede Neural quando usado a API himem com diferentes números de blocos reservados



Legenda conforme seção 5.4.1.
Fonte: o Autor, 2023.

Figura 15 – Tempo de execução da FFT em alguns dos ensaios realizados



Legenda conforme seção 5.4.1.
As retas estão acompanhadas da declividade obtida pela regressão linear de cada ensaio.

Fonte: o Autor, 2023.

6 Conclusões

Durante a realização do trabalho foram encontrados algumas adversidades, como um *bug* no alocador da esp-psram o qual necessitou ser corrigido durante o desenvolvimento da implementação. Também, após leitura das erratas do esp32 pode-se remover alguns gargalos de desempenho que são necessários apenas para revisões mais antigas do SoC. Apesar do grande poder do framework esp-idf, o mesmo conta com centenas de pull-requests abertos há meses no repositório oficial da biblioteca. O suporte, portanto, é limitado. O custo, porém, aparenta compensar para muitas aplicações.

6.1 Potencial de Aplicação

Apesar dos percalços, foi possível realizar com sucesso a execução do projeto pretendido, tendo por resultado tempos de execução inferiores aos obtidos nos testes preliminares.

O menor tempo de classificação por segmento de 9984 amostras, que representam 1,248 segundo, foi de 5,210 $\pm$ 0,003 segundos. Isso representa um tempo de processamento mais de quatro vezes o tempo de coleta da amostra, impedindo o processamento estritamente em tempo-real.

Entretanto, a sistemática de aplicação não necessita de processamento de todo o sinal em tempo-real. Mantendo o mesmo sistemo desenvolvido, é possível realizar contagem de mosquitos por amostragem temporal. A intenção do sistema não é necessariamente aferir o número total de mosquitos transitando por uma determinada região no espaço, mas sim permitir que os locais de maior incidência sejam conhecidos, e a incidência seja comparada com outros fatores ambientais e temporais.

Também, é plausível acumular áudio diariamente nos horários de maior atividade dos mosquitos, visto que a atividade é dependente do seu ciclo circadiano (GENTILE *et al.*, 2009; TAYLOR; JONES, 1969) e não é constante ao longo do dia e da noite. Assim, pode-se posteriormente processar em momentos de menor atividade esperada.

6.2 Trabalhos Futuros

Uma alternativa que propõe para estudos futuros é encontrar formas de reduzir a quantidade de áudio processada pelo classificador. Há de se considerar a possibilidade de haver um algoritmo intermediário com alta revocação (ou *recall*) funcionando como filtro

para descartar segmentos de baixa probabilidade de detecção antes de ocupar um longo tempo de CPU.

Apesar da linha ESP32 ter sido a linha do SoC disponível encontrado para este trabalho, a Espressif desenvolveu atualizações, agora comercializando a linha ESP32-S3 Series como substituta com desempenho melhorado. Sendo o produto mais novo, a posição de preço dos módulos ESP32-S3, em março de 2023, é superior à dos correspondentes ESP32 Series. Há potencial de se encontrar tempos de execução menores com essa nova linha em pesquisas posteriores.

Para outros trabalhos posteriores, há a possibilidade de explorar a quantização da rede neural. Através da quantização, pode-se ter uma rede neural com parâmetros int8 (inteiros de 8 bits) no lugar de float32 (ponto flutuante de precisão simples). De início, já espera redução de cerca de 75% da quantidade de memória utilizada para armazenar os parâmetros. Pela natureza das operações com inteiros ser mais simples, também há potencial de redução no tempo de execução e consumo de energia. Ao se efetuar alterações, deve-se reavaliar a qualidade da rede neural, observando a manutenção das suas taxas de acerto e outras métricas de desempenho.

Referências Bibliográficas

ALMEIDA, L. The fractional fourier transform and time-frequency representations. *IEEE Transactions on Signal Processing*, v. 42, n. 11, p. 3084–3091, 1994.

BOUOTO, A.; FREDDY, Y. Mosquitobell2: A mosquito detection system through the timbre attribute of the wing-beat frequency using an acoustic sensor. In: *2020 35th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. [S.l.: s.n.], 2020. p. 187–192.

CALVET, G.; AGUIAR, R. S.; MELO, A. S. O.; SAMPAIO, S. A.; FILIPPIS, I. de; FABRI, A.; ARAUJO, E. S. M.; SEQUEIRA, P. C. de; MENDONÇA, M. C. L. de; OLIVEIRA, L. de; TSCHOEKE, D. A.; SCHRAGO, C. G.; THOMPSON, F. L.; BRASIL, P.; SANTOS, F. B. dos; NOGUEIRA, R. M. R.; TANURI, A.; FILIPPIS, A. M. B. de. Detection and sequencing of zika virus from amniotic fluid of fetuses with microcephaly in brazil: a case study. *The Lancet Infectious Diseases*, Elsevier, v. 16, n. 6, p. 653–660, Jun 2016. ISSN 1473-3099. Disponível em: <https://doi.org/10.1016/S1473-3099(16)00095-5>.

CONLIN, R. *keras2c Readme*. 2022. Disponível em: <https://github.com/f0uriest/keras2c#readme>.

COOLEY, J.; TUKEY, J. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, v. 19, n. 90, p. 297–301, 1965. Disponível em: <https://doi.org/10.2307/2003354>.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, v. 2, n. 4, p. 303–314, dez. 1989. ISSN 1435-568X. Disponível em: <https://doi.org/10.1007/BF02551274>.

DAL CASTEL, L. K. *Fix Advanced Activation Layers Bugs and Create New Test 16*. 2023. Disponível em: <https://github.com/f0uriest/keras2c/pull/16>.

DONNELLE, D.; RUST, B. The fast fourier transform for experimentalists. part i. concepts. *Computing in Science Engineering*, v. 7, n. 2, p. 80–88, 2005.

EDER, M.; CORTES, F.; FILHA, N. Teixeira de S.; FRANçA, G. V. Araújo de; DEGROOTE, S.; BRAGA, C.; RIDDE, V.; MARTELLI, C. M. T. Scoping review on vector-borne diseases in urban areas: transmission dynamics, vectorial capacity and co-infection. *Infectious Diseases of Poverty*, v. 7, n. 1, p. 90, set. 2018. ISSN 2049-9957. Disponível em: <https://doi.org/10.1186/s40249-018-0475-7>.

ESPRESSIF. *ESP32 Datasheet*. 2022. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>.

ESPRESSIF. *ESP-IDF Programming Guide*. 2023. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html>.

ESPRESSIF. *ESP32 Series SoC Errata*. 2023. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_errata_en.pdf>.

ESPRESSIF. *Espressif IoT Development Framework*. 2023. Disponível em: <https://github.com/espressif/esp-idf>.

ESPRESSIF. *FreeRTOS (ESP-IDF)*. 2023. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/release-v5.0/esp32/api-reference/system/freertos_idf.html>.

ESPRESSIF. *The himem allocation API*. 2023. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/himem.html>.

FERNANDES, M. S.; CORDEIRO, W. *Detecting Aedes Aegypti Mosquitoes through Audio Classification with Convolutional Neural Networks*. 2020. Disponível em: <https://github.com/ComputerNetworks-UFRGS/CNN_Aedes_Aegypti_Classifier>.

FERNANDES, M. S.; CORDEIRO, W.; RECAMONDE-MENDOZA, M. Detecting aedes aegypti mosquitoes through audio classification with convolutional neural networks. *Computers in biology and medicine*, v. 129, 2021. Disponível em: <https://doi.org/10.1016/j.compbiomed.2020.104152>.

GARCEZ, P. P.; LOIOLA, E. C.; COSTA, R. M. D.; HIGA, L. M.; TRINDADE, P.; DELVECCHIO, R.; NASCIMENTO, J. M.; BRINDEIRO, R.; TANURI, A.; REHEN, S. K. Zika virus: Zika virus impairs growth in human neurospheres and brain organoids. *Science*, v. 352, n. 6287, p. 816 – 818, 2016. Cited by: 761. Disponível em: <https://www.science.org/doi/10.1126/science.aaf6116>.

GENTILE, C.; RIVAS, G. B. S.; MEIRELES-FILHO, A. C. A.; LIMA, J. B. P.; PEIXOTO, A. A. Circadian Expression of Clock Genes in Two Mosquito Disease Vectors: *cry2* Is Different. *Journal of Biological Rhythms*, v. 24, n. 6, p. 444–451, dez. 2009. ISSN 0748-7304, 1552-4531. Disponível em: <http://journals.sagepub.com/doi/10.1177/0748730409349169>.

GIBSON, G.; RUSSELL, I. Flying in tune: Sexual recognition in mosquitoes. *Current Biology*, Elsevier, v. 16, n. 13, p. 1311–1316, Jul 2006. ISSN 0960-9822. Disponível em: <https://doi.org/10.1016/j.cub.2006.05.053>.

GOOGLE. *Reduce TensorFlow Lite binary size*. 2022. Disponível em: <https://www.tensorflow.org/lite/guide/reduce_binary_size>.

GOOGLE. *TensorFlow Core | Machine Learning for Beginners and Experts*. 2022. Disponível em: <https://www.tensorflow.org/overview>.

GOOGLE. *TensorFlow Lite | ML for Mobile and Edge Devices*. 2022. Disponível em: <https://www.tensorflow.org/lite>.

HARRIS, F. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, v. 66, n. 1, p. 51–83, Jan 1978. ISSN 1558-2256.

HAYKIN, S. S.; VEEN, B. V. *Signals and Systems - 2nd ed.* [S.l.]: Wiley, 2002. 548 p. ISBN 0-471-16474-7.

KAHLIG, P. Some aspects of julius von hann's contribution to modern climatology. *Washington DC American Geophysical Union Geophysical Monograph Series*, v. 75, p. 1–7, 01 1993.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, May 2015. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/nature14539>.

MCFEE, B. *Use zero-padding by default most of the time 1382*. 2021. Disponível em: <https://github.com/librosa/librosa/pull/1382>.

MLAKAR, J. e. A. Zika virus associated with microcephaly. *New England Journal of Medicine*, v. 374, n. 10, p. 951–958, 2016. PMID: 26862926. Disponível em: <https://doi.org/10.1056/NEJMoa1600651>.

MUKUNDARAJAN, H.; HOL, F. J. H.; CASTILLO, E. A.; NEWBY, C.; PRAKASH, M. Using mobile phones as acoustic sensors for high-throughput mosquito surveillance. *eLife*, v. 6, 2017. Cited By :50. Disponível em: <https://doi.org/10.7554/eLife.27854>.

MURILLO, D.; MURILLO, A.; LEE, S. The Role of Vertical Transmission in the Control of Dengue Fever. *International Journal of Environmental Research and Public Health*, v. 16, n. 5, p. 803, mar. 2019. ISSN 1661-7827. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6427266/>.

NATIONAL INSTRUMENTS. *Understanding FFTs and Windowing*. 2021. Disponível em: <https://www.ni.com/en-il/innovations/white-papers/06/understanding-ffts-and-windowing.html>.

OLIVEIRA, N. *Aedes aegypti: conheça a história do mosquito no Brasil e suas características*. 2015. Disponível em: <https://agenciabrasil.ebc.com.br/geral/noticia/2015-12/aedes-aegypti-conheca-historia-do-mosquito-no-brasil-e-suas-caracteristicas>.

OMS. *A global brief on vector-borne diseases*. [S.l.], 2014. 54 p. p.

PORTNOFF, M. Time-frequency representation of digital signals and systems based on short-time fourier analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 28, n. 1, p. 55–69, 1980.

RITA, A. B.; FREITAS, R.; NOGUEIRA, R. M. R. *Dengue*. 2013. Disponível em: <https://agencia.fiocruz.br/dengue-0>.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, v. 323, n. 6088, p. 533–536, Oct 1986. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/323533a0>.

SLANEY, M. *Auditory Toolbox*. 1998. Disponível em: <https://engineering.purdue.edu/~malcolm/interval/1998-010/>.

SMS PORTO ALEGRE. *Petrópolis recebe armadilhas de monitoramento de mosquitos*. 2016. Disponível em: <http://www2.portoalegre.rs.gov.br/sms/default.php?p_noticia=183993&PETROPOLIS+RECEBE+ARMADILHAS+DE+MONITORAMENTO+DE+MOSQUITOS>.

STEVENS, S. S.; VOLKMANN, J.; NEWMAN, E. B. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, v. 8, n. 3, p. 185–190, 1937. Disponível em: <https://doi.org/10.1121/1.1915893>.

TAYLOR, B.; JONES, M. D. R. The circadian rhythm of flight activity in the mosquito Aedes aegypti (L.): the phase-setting effects of light-on and light-off. *Journal of Experimental Biology*, v. 51, n. 1, p. 59–70, 1969. Publisher: The Company of Biologists Ltd.

UNIVERSITY OF CAMBRIDGE. *Hidden Markov Toolkit*. 2016. Disponível em: <https://htk.eng.cam.ac.uk/>.

VASCONCELOS, D.; YIN, M. S.; WETJEN, F.; HERBST, A.; ZIEMER, T.; FöRSTER, A.; BARKOWSKY, T.; NUNES, N.; HADDAWY, P. Counting mosquitoes in the wild: An internet of things approach. In: *Proceedings of the Conference on Information Technology for Social Good*. New York, NY, USA: Association for Computing Machinery, 2021. (GoodIT '21), p. 43–48. ISBN 9781450384780. Disponível em: <https://doi.org/10.1145/3462203.3475914>.

WILSON, A. L.; COURTENAY, O.; KELLY-HOPE, L. A.; SCOTT, T. W.; TAKKEN, W.; TORR, S. J.; LINDSAY, S. W. The importance of vector control for the control and elimination of vector-borne diseases. *PLOS Neglected Tropical Diseases*, Public Library of Science, v. 14, n. 1, p. 1–31, 01 2020. Disponível em: <https://doi.org/10.1371/journal.pntd.0007831>.

Apêndices

APÊNDICE A – Dados de tempo de execução coletados nos ensaios

Tabela 8 – Tempo de execução

Freq. RAM	Correção de Cache	Freq. CPU	Blocos hi-mem	Precisão FFT	Compilação	Tempo (ms) Característica	Tempo (ms) Rede Neural	Tempo (ms) Total
40	memw	160	4	float	-O2	693 ± 6	9180 ± 3	9873 ± 6
40	memw	160	12	float	-O2	700 ± 3	7860 ± 3	8560 ± 3
40	memw	160	33	float	-O2	700 ± 3	7497 ± 6	8197 ± 6
40	memw	160	4	float	-Os	713 ± 6	14120 ± 3	14833 ± 6
40	memw	160	12	float	-Os	710 ± 3	12910 ± 3	13620 ± 3
40	memw	160	33	float	-Os	713 ± 6	12510 ± 3	13223 ± 6
40	memw	160	4	double	-O2	2290 ± 3	9180 ± 3	11470 ± 3
40	memw	160	12	double	-O2	2270 ± 3	7860 ± 3	10130 ± 3
40	memw	160	33	double	-O2	2270 ± 3	7480 ± 3	9750 ± 3
40	memw	160	4	double	-Os	2360 ± 3	14130 ± 3	16490 ± 3
40	memw	160	12	double	-Os	2340 ± 3	12920 ± 3	15260 ± 3
40	memw	160	33	double	-Os	2340 ± 3	12520 ± 3	14860 ± 3
80	não	240	4	float	-O2	460 ± 3	6200 ± 3	6660 ± 3
80	não	240	12	float	-O2	460 ± 3	5160 ± 3	5620 ± 3
80	não	240	33	float	-O2	460 ± 3	4750 ± 3	5210 ± 3
80	não	240	4	double	-O2	1950 ± 3	6200 ± 3	8150 ± 3
80	não	240	12	double	-O2	1950 ± 3	5110 ± 3	7060 ± 3
80	não	240	33	double	-O2	1950 ± 3	4760 ± 3	6710 ± 3
80	não	240	4	float	-Os	490 ± 3	10070 ± 3	10560 ± 3
80	não	240	12	float	-Os	490 ± 3	8990 ± 3	9480 ± 3
80	não	240	33	float	-Os	490 ± 3	8660 ± 3	9150 ± 3
80	não	240	4	double	-Os	1980 ± 3	10070 ± 3	12050 ± 3
80	não	240	12	double	-Os	1980 ± 3	8990 ± 3	10970 ± 3
80	não	240	33	double	-Os	1980 ± 3	8660 ± 3	10640 ± 3

Fonte: o Autor, 2023.