UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

LEOMAR MATEUS RADKE

AVALIAÇÃO DE MODELOS OTIMIZADOS DE *TINYML* PARA DETECÇÃO DE ANOMALIAS EM IOT

Porto Alegre
2022

LEOMAR MATEUS RADKE

AVALIAÇÃO DE MODELOS OTIMIZADOS DE *TINYML* PARA DETECÇÃO DE ANOMALIAS EM IOT

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Controle e Automação

ORIENTADOR: Prof. Dr. Ivan Müller

Porto Alegre
2022

LEOMAR MATEUS RADKE

AVALIAÇÃO DE MODELOS OTIMIZADOS DE *TINYML* PARA DETECÇÃO DE ANOMALIAS EM IOT

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Ivan Müller, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Banca Examinadora:

Prof. Dr. Ivanovitch Silva, UFRN
Doutor pela Universidade Federal do Rio Grande do Norte

Prof. Dr. Cleonilson Protásio Souza, UFCG
Doutor pela Universidade Federal de Campina Grande

Prof. Dr. Valner João Brusamarello, UFSC
Doutor pela Universidade Federal de Santa Catarina

Coordenador do PPGEE: _____
Sérgio Luís Haffner

Porto Alegre, outubro de 2022.

AGRADECIMENTOS

Ao Programa de Pós-Graduação em Engenharia Elétrica (PPGEE), pela oportunidade de realização de trabalhos em minha área de pesquisa.

Ao Professor Ivan Müller pelo apoio, incentivo e oportunidades de desenvolvimento dentro do grupo de pesquisa.

Ao meu grande amigo Max Feldman, companheiro nessa longa jornada, certamente esse trabalho não teria o acabamento necessário sem a tua colaboração.

À minha antiga empresa Falker, que flexibilizou a minha jornada no início do Mestrado.

À toda minha família, em especial aos meus pais e irmã, pelo suporte incondicional.

Agradeço minha companheira Patricia por dar o suporte em momentos difíceis e ser meu porto seguro.

RESUMO

O avanço das aplicações com *Internet of Things* (IoT), no contexto de redes de longa distância de baixa potência nos dias atuais é notório. Porém, algumas fragilidades ainda devem ser resolvidas, tais como a segurança dos dados trafegados, largura de banda utilizada e autonomia de bateria dos dispositivos. Este trabalho apresenta uma avaliação de modelos otimizados de *Tiny Machine Learning* (*TinyML*), onde são investigados os benefícios de se ter em um dispositivo sensor, um algoritmo otimizado onde a inferência dos dados é realizada localmente. O desempenho de cada uma das técnicas é avaliado, bem como a capacidade de redução que elas promovem. Um estudo de caso é apresentado em uma rede *LoRa*, onde um conjunto de dados é utilizado para avaliar o desempenho energético do modelo. O resultado evidencia redução de quase 4 vezes no consumo de energia na proposta de detecção de anomalia na borda.

Palavras-chave: Otimização, *TinyML*, Detecção de Anomalia, Redes de Longo Alcance e Baixa Potência.

ABSTRACT

The advancement of Internet of Things (IoT) applications in the context of low-power long-distance networks today is notorious. However, thus some weaknesses also appeared, such as the security of the data transmitted, bandwidth and battery life of the devices. This work presents an evaluation of optimized Tiny Machine Learning (TinyML) models. The benefits of having an optimized algorithm in a sensor device are evaluated, where the data inference is performed locally. The performance of each of the techniques will be evaluated, as well as the reduction capacity they promote. A case study is presented in a LoRa network, where a dataset is used to evaluate the energy performance of the model. The result was an approximate 4x drop in power consumption in the edge anomaly detection.

Keywords: Optimization, *TinyML*, Anomaly Detection, Low-Power Wide-Area Network.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CNN	*Convulational Neural Network*
CPU	*Central Processing Unit*
HW	*Hardware*
IA	*Inteligência Artificial*
IoT	*Internet of Things*
kNN	*k-Neighbors Nearest*
$LPWAN$	*Low Power Wide Area Network*
$M2M$	*Machine-to-machine*
MAE	*Mean Absolute error*
MCU	*Microcontroller Unit*
ML	*Machine Learning*
MSE	*Mean Square Error*
PC	*Personal Computer*
PCA	*Principal Component Analysis*
QAT	*Quantization Aware-Training*
QPT	*Quantization Post-Training*
RAM	*Random Access Memory*
RMS	*Root Mean Square*
RNA	*Rede Neural Artificial*
RTC	*Real-Time Clock*
SPI	*Serial Peripheral Interface*
SW	*Software*

LISTA DE SÍMBOLOS

σ	*Desvio Padrão*
μ	*Média*
T	*Threshold*

SUMÁRIO

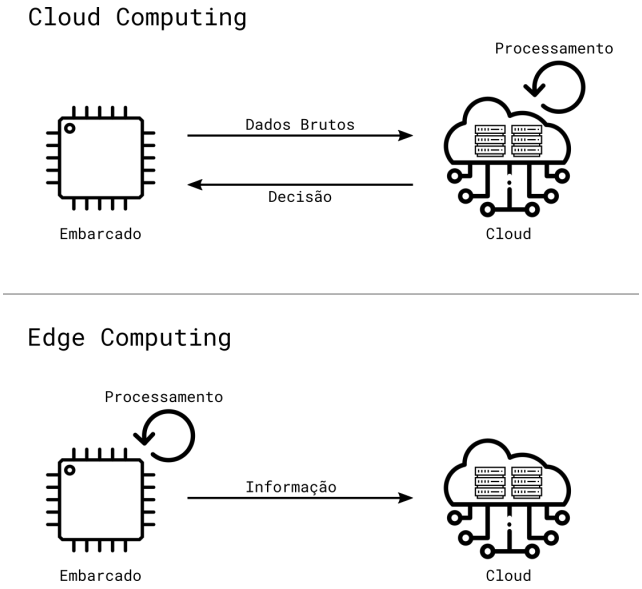
1 INTRODUÇÃO

Atualmente há uma demanda crescente por aplicativos e tecnologias de IoT em todo o mundo. Prevê-se que os dispositivos em rede aumentarão de 18 bilhões em 2017 para 28,5 bilhões em 2022, e as conexões *Machine-to-Machine* (M2M) atingirão 15 bilhões em 2022 (CISCO, 2018). Dentro da IoT, uma das aplicações mais difundidas está na coleta de dados em dispositivos sensores para um posterior envio a uma central, que irá processar, transformar e extrair alguma informação acerca do dado disponível.

Esse processo apresenta algumas fragilidades e pode até levar a ineficiência da operação. Por exemplo, ter dados de uma aplicação, em muitos casos sensíveis, trafegando até um nó concentrador ou *gateway* é algo problemático, principalmente quando levados em consideração os aspectos de limitações de banda e bateria dos dispositivos finais da rede. Algumas soluções que visam mitigar esses problemas estão concentradas em arquiteturas e protocolos de rede, como pode ser visto em Redes de Longa Distância de Baixa Potência, traduzido de *Low Power Wide Area Network* (LPWAN).

Esse tipo de especificação, que engloba uma série de protocolos e mecanismos acabaram trazendo um novo conceito para a IoT: a computação de borda (SARKER *et al.*, 2019). Os dados coletados são processados, transformados e inferidos no dispositivo sensor, evitando tráfego desnecessários de dados, reduzindo consumo de bateria na transmissão e trazendo maior segurança, visto que os dados da aplicação podem não ser transmitidos. A Figura 1 apresenta uma simplificação das duas possibilidades de uma aplicação ser executada. Na parte superior, está uma representação da utilização de uma infraestrutura *cloud*, onde o processamento está na nuvem. Na parte inferior, encontra-se a solução *edge*, que se baseia em levar o processamento para a borda da aplicação.

Figura 1 – Simplificação das soluções *cloud* e *edge computing*.



Fonte: Autor.

Ao utilizar uma solução de borda, será necessária uma abordagem que leve em conta as recursos disponíveis. Em dispositivos com baixo poder computacional, os algoritmos de Inteligência Artificial (IA) necessitam de adaptações e técnicas de otimização, para que os modelos possam ser armazenados em poucos KBytes e possam ser executados com pouca memória RAM. O *Machine Learning* (ML) é outro bloco de construção essencial na IoT. Devido à ampla utilização e ao consumo mínimo de energia dos dispositivos IoT, há cada vez mais interesse em fornecer funcionalidades IA na borda. Os recentes avanços em IA são impulsionados por dados massivos, computação de borda e consumo de energia (REN; ANICIC; RUNKLER, 2021).

Para preencher a lacuna entre microcontroladores e ML, especialmente no aprendizado profundo, o *TinyML* dedica-se a oferecer soluções baseadas em Redes Neurais Artificiais (RNA) na borda. Resultados notáveis foram produzidos recentemente, como reconhecimento de voz (SANCHEZ-IBORRA; SKARMETA, 2020) e previsões de pressão atmosférica (ALONGI *et al.*, 2020). As principais vantagens do *TinyML* estão relacionadas a:

- Privacidade: os dados são processados na borda, enquanto a transmissão de dados para a nuvem pode violar as políticas de privacidade e é vulnerável à interceptação.
- Latência: Todo o processo, que acontece na borda, é independente da comunicação

externa. Assim, os dispositivos de borda podem tomar decisões em tempo real.

- Eficiência energética: A rede neural consome muita energia, mas a transmissão dos dados para a nuvem precisa de uma ordem de magnitude maior de energia.

1.1 Motivação

Como a evolução das rede IoT e de computação em nuvem, alguns gargalos foram detectados em ambos os conceitos. Ter todos os dados sensíveis de uma aplicação trafegando, bem como uma determinada limitação de banda e bateria dos dispositivos finais das redes, acabaram trazendo um novo conceito para a IoT: a computação em borda, ou seja, o processamento dos dados coletados são tratados no dispositivo de campo. Entretanto, essa tarefa pode ser extremamente complexa, demandando muitos recursos de hardware, inviabilizando determinadas aplicações.

Para tal situação, apresentam-se os algoritmos de aprendizado de máquina. Na utilização para processamento em borda, esses algoritmos normalmente são acompanhados de técnicas de redução dos modelos, adequando assim a realidade dos recursos de CPU e memória dos dispositivos.

1.2 Objetivos

O objetivo geral do presente trabalho é a elaboração de um sistema de processamento de inferência na borda de uma LPWAN, avaliando modelos otimizados para hardware reduzidos, trazendo menor latência na decisão, maior segurança e menor consumo de energia.

Os objetivos específicos deste trabalho são:

- Comparar o consumo de energia dos dispositivos de rede. Para tal, são testados os dois modos de operação, *cloud* e borda, com os mesmos dispositivos.

- Fazer uma comparação entre o modelo completo de redes neurais artificiais e os modelos otimizados, que são ideais para dispositivos com poucos recursos computacionais.

Este trabalho está organizado da seguinte forma: no Capítulo 2, são apresentados os conceitos básicos utilizados ao longo da dissertação. No Capítulo 3 é apresentada

uma revisão bibliográfica referente a trabalhos anteriormente realizados. Os métodos e materiais utilizados são apresentados no Capítulo 4. Já a implementação do sistema proposto é demostrado no Capítulo 5. No Capítulo 6 são relatados os resultados obtidos durante a realização dos estudos de caso. Por fim, conclusões e trabalhos futuros são apresentados no Capítulo 7.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica necessária para as avaliações dos modelos de quantização para aprendizado de máquina em computação de borda para IoT. A organização do capítulo é feita da seguinte maneira: Na seção 2.1 é apresentado o conceito de IoT. Na seção 2.2 é apresentado de forma resumida um panorama das redes de longa distância. Uma visão geral da *Edge Computing* é apresentada na seção 2.3. A seção 2.4 apresenta o conceito de *TinyML*. A seção 2.5 apresenta todas as técnicas de otimização utilizadas neste trabalho. Um breve resumo de detecção de anomalias é apresentado na seção 2.6. e, por fim, a seção 2.7 apresenta métricas de desempenho utilizadas.

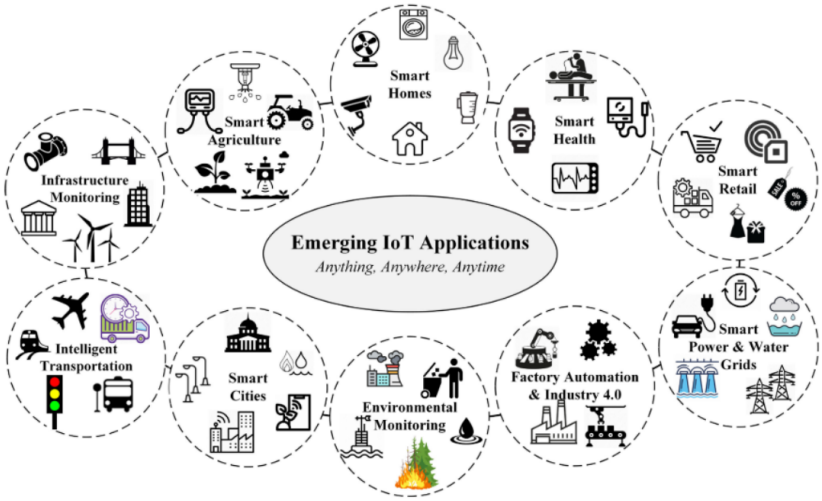
2.1 Internet das Coisas

Internet of Things (IoT), termo em inglês para "Internet das Coisas", passou a representar dispositivos elétricos ou eletrônicos, de capacidades e tamanhos variados, que estão de alguma forma conectados à Internet. O escopo das conexões está sempre se ampliando para além da comunicação M2M. Os dispositivos IoT empregam uma ampla gama de protocolos de rede, aplicações e domínios de rede (MULLIGAN, 2014). IoT envolve o desenvolvimento de uma rede inteligente que pode ser controlada e programada. Os dispositivos utilizados nesta rede utilizam a tecnologia embarcada, que lhes permite comunicar-se direta ou indiretamente utilizando a infraestrutura da Internet (CHASE, 2013).

Muitas aplicações de IoT, como ilustra a Figura 2, consistem em monitoramento: medição de rede elétrica ou rede de distribuição de água, monitoramento de nível de bateria de veículo elétrico, monitoramento meteorológico, de temperatura, umidade, entre outras. Uma grande quantidade de informações, das mais diversas aplicações estão sendo disponibilizadas para as mais diversas áreas. Isso, no entanto, traz alguns pontos de atenção,

entre eles, a segurança da informação. Um volume considerável de dados está trafegando em redes que, na maioria das vezes, poderiam ser processados localmente ao invés de serem enviados a uma central.

Figura 2 – Gama de aplicações IoT



Fonte: (SWAMY; KOTA, 2020)

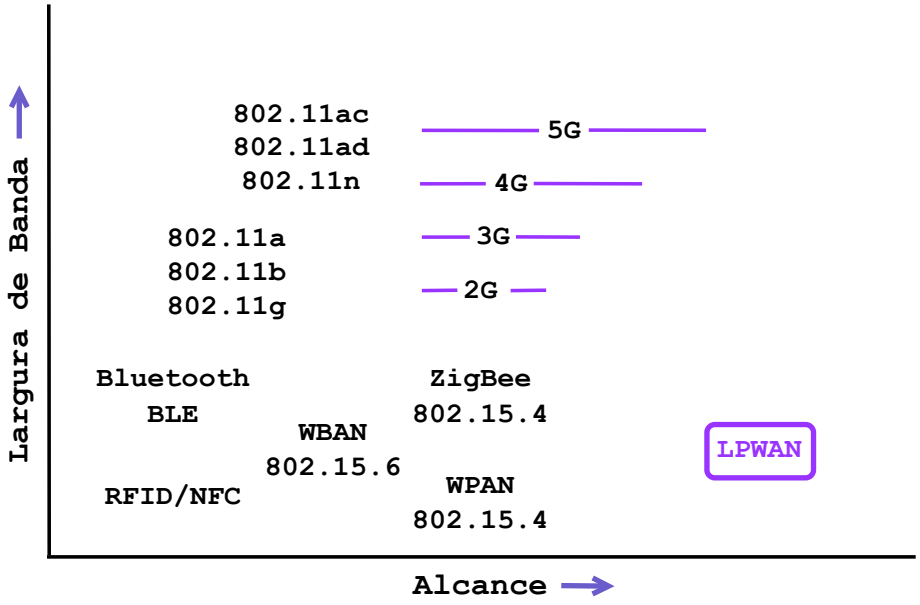
2.2 *Low Power Wide Area Network*

Tecnologias LPWAN representam um novo paradigma de comunicação, que complementa as tradicionais redes sem fio de curto alcance - Zigbee/IEEE 802.15.4 em redes de sensores sem fio, Bluetooth, identificação por radiofrequência (RFID), afim de atender a diversos requisitos de aplicativos de IoT (FADLULLAH *et al.*, 2011). Na Figura 3 é possível observar como a LPWAN está posicionada em relação a outros protocolos, quando as características de largura de banda e alcance são avaliadas.

As tecnologias LPWAN oferecem conjuntos exclusivos de recursos, incluindo conectividade de área ampla para dispositivos de baixa potência e baixa taxa de dados. Devido à limitação dos sistemas de comunicação de curto alcance, a tecnologia LPWAN foi projetada especificamente com os objetivos de baixo consumo de energia e ampla cobertura. Em particular, esquemas de modulação de banda ultra-estreita e *Chirp Spread Spectrum* foram propostos para a camada física de sistemas LPWAN em comunicação M2M graças ao seu excelente desempenho de cobertura e baixo consumo.

Para possibilitar baixo consumo de energia, a topologia em estrela e o método de acesso aleatório são empregados na camada *Media Access Control* (MAC). Uma carac-

Figura 3 – Visão de alcance X largura de banda de alguns protocolos de rede.



Fonte: Autor.

terística fundamental dessas técnicas LPWAN é que elas fornecem compensação entre a taxa de dados, vida útil da bateria e custos de implantação. Essa compensação é aceitável para a maioria dos aplicativos M2M, que não exigem altas taxas de dados e baixa latência (XIONG *et al.*, 2015).

2.3 *Edge Computing*

Para atender aos requisitos computacionais do aprendizado profundo, uma abordagem comum é aproveitar a computação em *cloud*. Para usar os recursos do *cloud*, os dados devem ser movidos do local da fonte de dados na borda da rede, por exemplo, de smartphones e sensores de IoT para um local centralizado na *cloud*. Essa solução, de se mover os dados da fonte para a nuvem, apresenta vários desafios, tais como (CHEN; RAN, 2019):

- Latência: a inferência em tempo real é fundamental para muitos aplicativos. Por exemplo, os quadros da câmera de um veículo autônomo precisam ser processados rapidamente para detectar e evitar obstáculos ou um aplicativo de assistência baseado em voz precisa analisar e entender rapidamente a consulta do usuário e retornar uma resposta. No entanto, o envio de dados para a *cloud* para inferência ou treina-

mento pode incorrer em atrasos adicionais de enfileiramento e propagação na rede e pode não atender os requisitos rigorosos de baixa latência fim-a-fim necessários para aplicativos interativos em tempo real;

- Escalabilidade: o envio de dados das fontes para a *cloud* apresenta problemas de escalabilidade, pois o acesso de rede pode se tornar um gargalo à medida que o número de dispositivos conectados aumenta. O *upload* de todos os dados para o *cloud* também é ineficiente em termos de utilização de recursos de rede, principalmente se nem todos os dados de todas as fontes forem necessários.

- Privacidade: o envio de dados para a *cloud* traz riscos à privacidade dos usuários que possuem os dados ou cujos comportamentos são capturados nos dados. Os usuários podem não confiar ao carregar suas informações confidenciais na *cloud*.

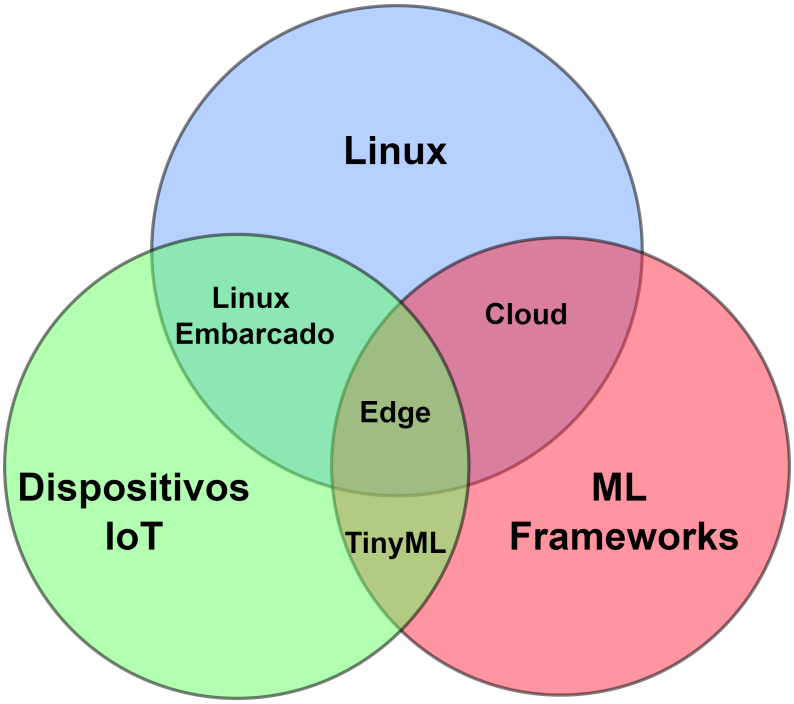
Edge computing é uma solução viável para atender aos desafios de latência, escalabilidade e privacidade, descritos anteriormente. Para enfrentar os desafios de latência, a computação de borda faz com que haja a proximidade dos dados coletados por sensores nos dispositivos terminais, diminuindo a latência de ponta a ponta e, portanto, permitindo serviços em tempo real. Para enfrentar os desafios de escalabilidade, a computação de borda permite uma arquitetura hierárquica de dispositivos finais, nós de computação de borda e *data centers* em *cloud* que podem fornecer recursos de computação e dimensionar com o número de clientes, evitando gargalos de rede em um local central. Para enfrentar os desafios de privacidade, a *edge computing* permite que os dados sejam analisados perto da fonte, talvez por um servidor de borda local confiável, evitando assim a propagação pela Internet e reduzindo a exposição a ataques de privacidade e segurança.

2.4 *TinyML*

TinyML é um conceito promissor que lida com a execução de modelos de ML otimizados para microprocessadores de ultrabaixa potência (< 1mW) com o mínimo consumo de energia (BANBURY *et al.*, 2020). Especialmente relevante para hardwares reduzidos de 8, 16 e 32 bits por exemplo, o TinyML é uma ferramenta valiosa para aprimorar os recursos de processamento, especialmente porque o processamento de dados e os serviços de ML no dispositivo aumentam as funcionalidades no próprio dispositivo.

A Figura 4 ilustra as áreas de tecnologia e facilitadores como círculos e seus pontos comuns como interseções. Nessa visão, por exemplo, o mundo do Linux Embarcado pode ser considerado um ponto de encontro entre os dispositivos Linux e IoT, reconhecendo também que os recursos do dispositivo IoT se estendem por aplicações *Edge*. O ML foi originalmente desenvolvido em máquinas *cloud*, com estruturas de software exigentes e com grandes recursos de hardware, como unidades de processamento gráfico poderosas. Agora, a computação está se movendo para a borda, executando ML em recursos menos poderosos, mas ainda com sistemas operacionais incorporados que o suportam. O *TinyML*, por sua vez, representa o ponto de conexão entre dispositivos IoT restritos e ML sem nenhum sistema operacional rico em recursos.

Figura 4 – Interseção de sistemas IoT incorporados a aprendizado de máquina.



Fonte: Adaptado de (DOYU; MORABITO, 2019).

Ao adotar o *TinyML*, cada dispositivo IoT se torna inteligente, ofertando a capacidade de analisar dados na borda, acelerando a tomada de decisões. Devido ao seu potencial para executar modelos de ML em um ambiente de recursos limitados e de baixo custo, o *TinyML* atraiu atenção da academia e da indústria. Especialistas em pesquisa de grandes empresas de tecnologia e acadêmicos convergiram e estão trabalhando para o desenvolvimento de soluções *TinyML*. Isso também levou ao início da comunidade *TinyML* em

2019, cujo objetivo é desenvolver sistemas, hardware, algoritmos, software e aplicativos para o *framework TinyML* (ML, 2019).

2.5 Otimização de Modelos de *Machine Learning*

Um importante aspecto dentro do *TinyML* reside na importância de que os modelos de ML possam ser alocados em dispositivos com poucos recursos de hardware. Nesse contexto, foram necessários estudos para desenvolver ferramentas de otimização dos modelos. Algumas dessas técnicas são elencadas a seguir.

2.5.1 Quantização Pós-Treino

A quantização pós-treino (QPT) é aquela em que se representam os vetores e pesos e ativações com menor precisão, como inteiros de 8 bits em oposição a $float$ de 32 bits, preservando vetores de *bias* em 32 bits. Um requisito básico do esquema de quantização é que ele permita a implementação eficiente de toda aritmética usando apenas operações aritméticas inteiras nos valores quantizados. Isso é equivalente a exigir que o esquema de quantização seja um mapeamento linear de inteiros q para números reais r, para algumas constantes S e Z (JACOB *et al.*, 2018). A equação (1) é o esquema de quantização e as constantes S e Z são parâmetros de quantização.

$$r = (q - Z) \times S \tag{1}$$

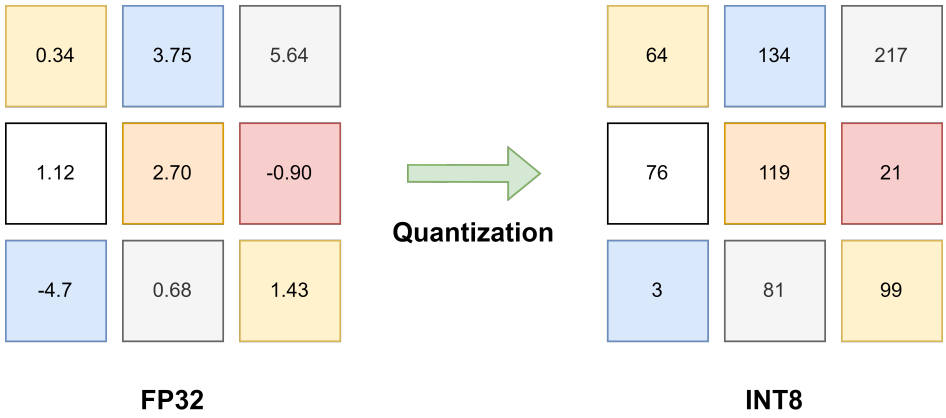
O esquema de quantização usa um único conjunto de parâmetros de quantização para todos os valores dentro de cada *array* de ativações e dentro de cada *array* de pesos. A constante S, de *scale*, é um número real positivo arbitrário. Normalmente é representado em software como uma quantidade de ponto flutuante, como os valores reais r. A constante Z, de *zero-point*, é do mesmo tipo que os valores quantizados q, e é de fato o valor quantizado q correspondente ao valor real 0. Isso permite atender automaticamente ao requisito de que o valor real $r = 0$ seja exatamente representável por um valor quantizado. A motivação para esse requisito é que a implementação eficiente de operadores de rede neural geralmente requer preenchimento zero de *arrays* em torno dos limites (JACOB *et al.*, 2018).

É importante destacar que todo o processo de criação do modelo quantizado está baseado em um modelo já treinado. Desta forma, é necessário fornecer uma amostra repre-

sentativa do conjunto de dados, pois ela irá permitir que o conversor estime uma faixa dinâmica para todos os dados variáveis. O conjunto de dados não precisa ser exclusivo em comparação com o conjunto de dados de treinamento ou avaliação.

Ao fornecer o conjunto de dados representativo ao conversor, o parâmetro de otimização realizará a quantização inteira no modelo de entrada. A Figura 5 ilustra uma matriz de pesos 32 bits que são quantizados para 8 bits.

Figura 5 – Exemplo de quantização de uma matriz de pesos de $float32$ para $uint8$.



Fonte: Adaptado de (NETA ZMORA; RODGE, 2021).

Os pesos de um modelo agora podem receber apenas um pequeno conjunto de valores e as diferenças mínimas entre eles são perdidas, acarretando em perda de informações. Por exemplo, todos os valores no intervalo [2.0, 2.3] agora podem ser representados em um único valor. Isso é semelhante ao efeito do arredondamento quando valores fracionários são representados como números inteiros.

Existem também outras fontes de perdas. Quando esses números com perdas são usados em vários cálculos de adição de multiplicação, essas perdas se acumulam. Além disso, os valores $uint8$, que se acumulam em inteiros $int32$, precisam ser redimensionados para valores $uint8$ para a próxima computação, introduzindo assim mais erros computacionais.

2.5.2 Treinamento com Reconhecimento de Quantização

A ideia central do *Quantization Aware Training* (QAT), treinamento com reconhecimento de quantização em português, está em simular a computação de tempo de inferência de baixa precisão na passagem direta do processo de treinamento. Isso introduz o erro de quantização como ruído durante o treinamento e como parte da perda geral, que o algo-

ritmo de otimização tenta minimizar. Assim, o modelo aprende parâmetros que são mais robustos à quantização.

Existem várias maneiras para realizar o QAT, desde começar com um modelo não treinado até começar com um modelo pré-treinado. Todas as formas alteram o regime de treinamento para incluir o erro de quantização na perda de treinamento, inserindo operações de quantização falsa no gráfico de treinamento para simular a quantização de dados e parâmetros. Essas operações são chamadas de falsas porque quantificam os dados, mas imediatamente "desquantizam" os mesmos para que a computação da operação permaneça com precisão de ponto flutuante. Essa condição adiciona ruído de quantização sem alterar muito a estrutura de aprendizado profundo (NETA ZMORA; RODGE, 2021).

A etapa de *backpropagation* ainda acontece com todos os pesos e *bias* armazenados em ponto flutuante para que possam ser facilmente alterados por pequenas quantidades. A passagem de propagação direta, no entanto, simula a inferência quantizada como acontecerá no mecanismo de inferência, implementando em aritmética de ponto flutuante o comportamento de arredondamento do esquema de quantização, seguindo o mesmo padrão apresentado na subseção 2.5.1.

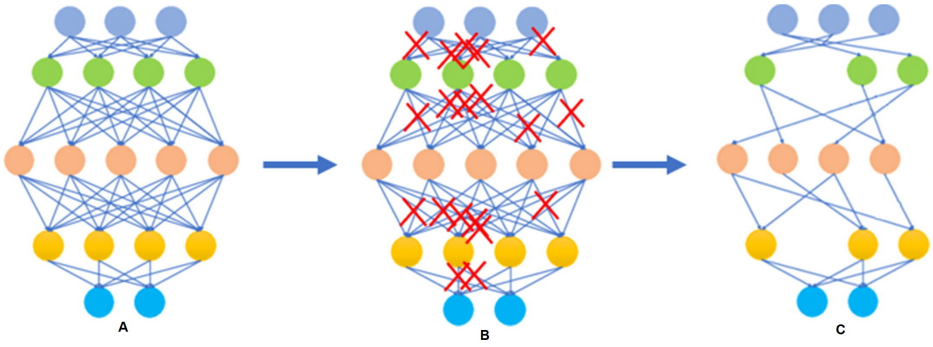
2.5.3 *Pruning*

O *pruning*, que significa poda em português, é a técnica de redução de um modelo de aprendizado de máquina que consiste em remover (definir como 0 permanentemente) determinados pesos. Normalmente os pesos podados são aqueles que já estão próximos de 0 (em valor absoluto). Isso impede um modelo de *overfitting*, pois os pesos que foram considerados inúteis no início do treinamento não podem ser reativados novamente. Uma representação gráfica do *pruning* é mostrada na Figura 6.

Para cada camada escolhida para ser podada, adicionamos uma variável de máscara binária que é do mesmo tamanho e forma que o tensor de peso da camada e determina quais dos pesos participam da execução direta da rede. São injetadas operações no treinamento para classificar os pesos nessa camada por seus valores absolutos para zerar os pesos de menor magnitude até que algum nível de esparsidade S desejado seja alcançado.

Os gradientes retropropagados fluem pelas máscaras binárias e os pesos que foram mascarados na execução direta não são atualizados na etapa de retropropagação. Introduz-se um novo algoritmo de poda gradual automatizado no qual a esparsidade é aumentada

Figura 6 – Rede neural completa em **A**, remoção de sinapses e neurônios em **B** resultando em uma nova RNA em **C**.



Fonte: Adaptado de (ZIV; GOLDBERGER; Riklin Raviv, 2021)

de um valor de esparsidade inicial S_0, geralmente 0, para um valor de esparsidade final S_f em um intervalo de n etapas de poda, começando na etapa de treinamento t_0 e com frequência de poda ΔT (ZHU; GUPTA, 2017). A equação (2) define a esparsidade S desejada.

$$S = (S_f - S_0)\left(\frac{t - t_0}{n\Delta T}\right)^3 \tag{2}$$

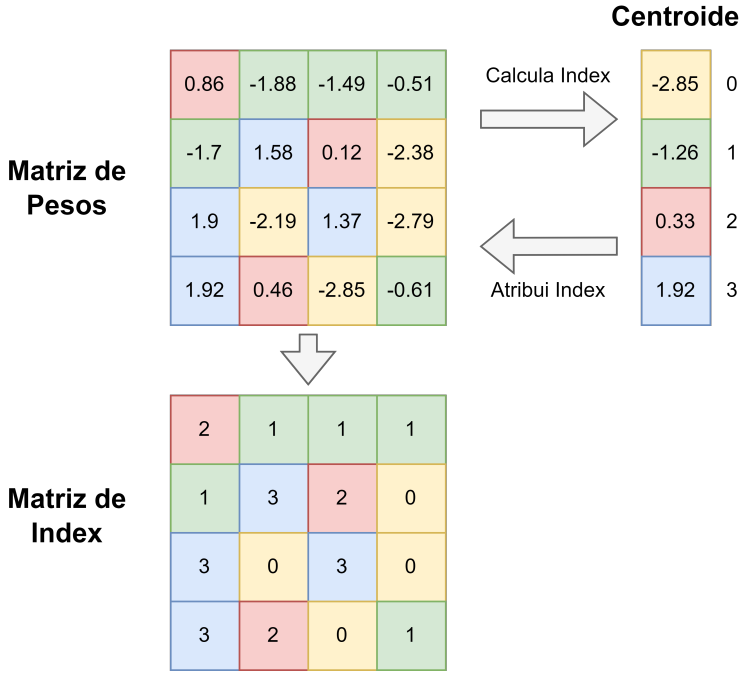
As máscaras de peso binário são atualizadas a cada ΔT etapas à medida que a rede é treinada para aumentar gradualmente a esparsidade da rede, permitindo que as etapas de treinamento da rede se recuperem de qualquer perda de precisão induzida pela poda. Foi verificado que variar a frequência de poda ΔT entre 100 e 1000 etapas de treinamento teve um impacto insignificante na qualidade final do modelo (ZHU; GUPTA, 2017).

2.5.4 Agrupamento de pesos

O *clustering*, ou agrupamento de pesos, reduz o número de valores de peso exclusivos em um modelo, gerando benefícios para a implantação. Ele primeiro agrupa os pesos de cada camada em N *clusters* e, em seguida, compartilha o valor do centroide do *cluster* para todos os pesos pertencentes ao *cluster*.

Na Figura 7 está uma exemplificação de como são agrupados os pesos. Uma camada do modelo contém uma matriz de pesos 4x4 (representada pela “matriz de pesos”). Cada peso é armazenado usando um valor float32. Ao salvar o modelo, está se armazenando 16 valores float32 exclusivos em disco.

Figura 7 – Agrupamento de pesos de uma camada de uma RNA.



Fonte: Adaptado de (ALAN CHIAO, 2020).

O agrupamento de pesos reduz o tamanho do modelo substituindo pesos semelhantes em uma camada com o mesmo valor. Esses valores são encontrados executando um algoritmo de agrupamento sobre os pesos treinados do modelo. O usuário pode especificar o número de *clusters*. Em seguida, cada peso na matriz de pesos é substituído pelo índice de seu centroide. Esta etapa é mostrada em "atribuir índices". Agora, em vez de armazenar a matriz de peso original, o algoritmo de agrupamento de peso pode armazenar a matriz modificada mostrada em "matriz de index" (contendo o índice dos valores do centroide) e os próprios valores do centroide. Neste caso, é reduzido o tamanho de 16 floats únicos para 4 floats e 16 índices de 2 bits. A economia aumenta com tamanhos de matriz maiores. Observe que, mesmo que ainda que fossem armazenados 16 floats, eles agora têm apenas 4 valores distintos. Ferramentas comuns de compactação - como gzip - agora podem aproveitar a redundância nos dados para obter uma compactação mais alta (ALAN CHIAO, 2020).

2.6 Detecção de Anomalia

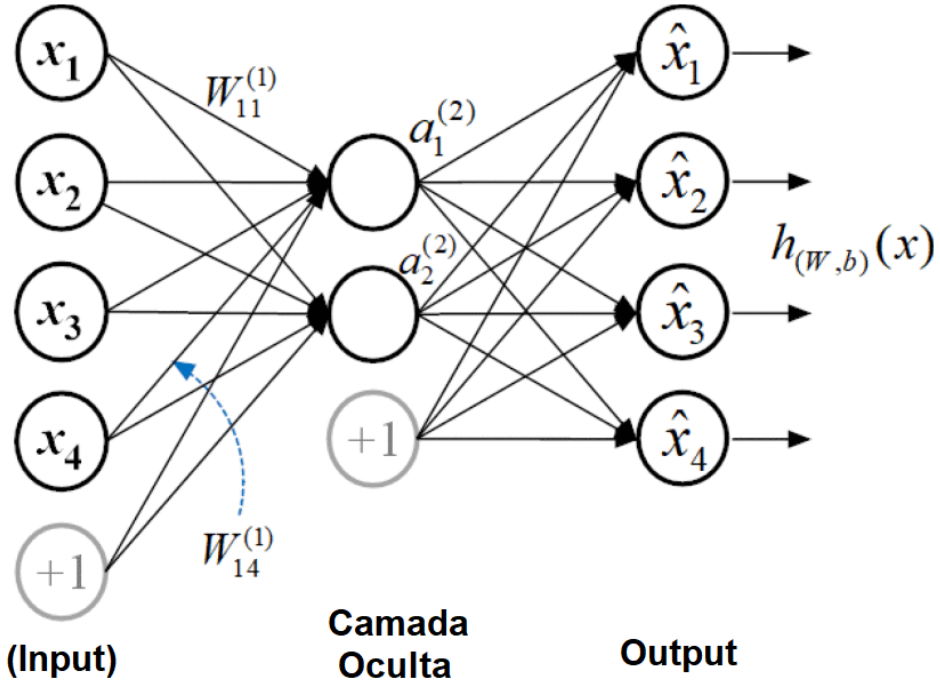
Detecção de anomalias referem-se à descoberta dos eventos/dados periféricos no conjunto de dados. Por exemplo, em um conjunto de dados de tráfego de servidor, as ano-

malias referem-se à intrusão ou qualquer atividade suspeita na rede, como um ataque de força bruta. No aprendizado de máquina, existem vários algoritmos e metodologias para identificar atividades suspeitas. Uma maneira de identificar anomalias é reconhecer pontos de dados que se desviam da maioria do padrão ou padrões normais (TOSHNIWAL; MAHESH; JAYASHREE, 2020). Um desses algoritmos de detecção de anomalias são as redes neurais artificias do tipo *autoencoder*.

2.6.1 Rede Neural Artificial *Autoencoder*

Uma rede neural artificial é um grupo interconectado de nós de processamento, ou seja, "neurônios", que realizam conjuntamente uma transformação (tipicamente não linear) de entradas em determinadas saídas desejadas. Um *autoencoder* é um tipo especial de redes neurais cujo objetivo é reconstruir as entradas em vez de prever algumas variáveis de destino. Ao reconstruir as entradas, um *autoencoder* tenta aprender uma representação condensada dos dados de entrada, um processo também conhecido como "codificação" (LUO; NAGARAJAN, 2018). Uma típica RNA *autoencoder* é mostrada na Figura 8.

Figura 8 – Exemplo de uma RNA *autoencoder*.



Fonte: Adaptado de (LUO; NAGARAJAN, 2018).

1. Camada de entrada: um vetor de dimensão M que representa os sinais de entrada, denotado por: **x** =(x_1, x_2, ..., x_M);

2. Camada de saída: denotada por vetor $\widehat{\mathbf{x}}$ = ($\widehat{x}_1$, $\widehat{x}_2$, ..., $\widehat{x}_M$). Observe que isso é diferente das redes neurais gerais, nas quais utilizam **y** =(y_1, y_2, ..., y_M) para denotar a camada de saída. No caso de *autoencoders*, a saída tem a mesma dimensão que a entrada, e é desejado que a saída seja igual à entrada para fins de reconstrução da entrada original.

3. Uma ou várias camadas ocultas: situadas entre as camadas de entrada e saída, essas camadas ocultas visam aprender um padrão nas entradas para "codificar" as informações essenciais. Denotando o número total de camadas por L e indexar as camadas por $l = 1, 2, \ldots, L$, e denotar o número de nós na l-ésima camada por $n^{(l)}$.

A capacidade de um *autoencoder* treinado reconstruir qualquer vetor de entrada fornece algumas informações sobre o quão normal é esse vetor de entrada. Um erro de reconstrução mais alto sugere que há alguma informação nos dados de entrada que não é esperada, tendo em vista os dados usados para treinar essa rede (COOK; MıSıRLı; FAN, 2020).

2.7 Análise de Desempenho

Avaliar o desempenho de sistemas de detecção de anomalias, algumas métricas são necessárias. Métricas gerais de desempenho do sistema não específicas do modelo de ML incluem taxa de transferência, latência e energia (CHEN; RAN, 2019). Para problemas de detecção de anomalias, com dados desbalanceados (maior ocorrência de uma classe em relação à outra), é necessário ter atenção à qual métrica se torna mais relevante para a aplicação, visto que um modelo trivial poderia ter uma taxa de acerto com valor aparentemente alto baseando-se apenas em um palpite constante da classe majoritária ou um palpite aleatório baseado na estatística base das classes. Desta forma, um breve contexto das principais métricas de avaliação de desempenho torna-se necessário.

2.7.1 *Precision, Recall* e *F1-Score*

Precisão avalia o quão exato é o método. Para tal, ela avalia a quantidade de resultados corretos em todo universo de resultados previstos como pertencentes à classe 1 (positivos). A precisão pode ser definida baseada na equação 5, como a razão entre a quantidade de resultados Verdadeiro Positivo (VP) e a soma de resultados verdadeiros e resultados

Falsos Positivos (FP). Sensibilidade, do inglês *Recall*, trata-se do número de exemplos classificados como pertencentes a uma classe, que realmente são daquela classe.

Inicialmente, é criada a matriz confusão (3) dos dados de teste. Os casos Verdadeiro Positivo (VP), Falso Positivo (FP), Falso Negativo (FN) e Verdadeiro Negativo (VN) fazem a composição da matriz.

$$M_{Confu} = \begin{bmatrix} VP & FP \\ FN & VN \end{bmatrix} \tag{3}$$

A partir disso, calcula-se a *Precision* (5) - onde o denominador é o somatório de condições positivas, a o *Recall* (4) - onde o denominador é o somatório das condições positivas previstas - para obtenção da $F1_{Score}$ (AHMAD *et al.*, 2019).

$$Recall = \frac{VP}{VP + FP} \tag{4}$$

$$Precision = \frac{VP}{VP + FN} \tag{5}$$

A *F1-Score* é uma maneira de visualizar as métricas de *Precision* e *Recall* juntas. A união das duas métricas poderia ser realizada através da média aritmética, contudo isso pode ser um problema quando tem-se valores baixos de algum dos índices. Como o F1 score é resultante da média harmônica entre *Precision* e *Recall*, como mostra a equação 6, seu resultado será dominado pela métrica de pior desempenho entre as duas (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$F1_{Score} = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)} \tag{6}$$

2.7.2 Acurácia

Acurácia é uma métrica de taxa de acerto. Em problemas com classes desproporcionais, ela causa uma falsa impressão de bom desempenho, devido ao modo de cálculo desta métrica. Um bom exemplo desse caso, em um *dataset* em que 80% dos rótulos pertencem a uma classe, é possível obter uma acurácia dessa ordem também, em torno de 80%, apenas retornando a saída como sendo a da classe majoritária. Contudo, embora tenha acurácia de 80%, o sistema seria incapaz de detectar os eventos da classe minoritária (GOODFELLOW; BENGIO; COURVILLE, 2016).

Mesmo assim, é importante a avaliação, para mensurar esse dados e comparar com as outras métricas de avaliação de desempenho. É calculada também a *Accuracy* (7), onde o denominador é o somatório de todas as amostras avaliadas.

$$Accuracy = \frac{(VP + VN)}{(VP + VN + FP + FN)} \tag{7}$$

3 ANÁLISE DO ESTADO DA ARTE

Neste capítulo é realizada a análise do estado da arte, onde são apresentados alguns dos principais trabalhos relacionados com o tema desta pesquisa. Na seção 3.1 são apresentados trabalhos relacionados à detecção de anomalias. A seção 3.2 apresenta trabalhos realizados em relação a utilização de *Edge Computing*. Por fim, a seção 3.3 apresenta trabalhos diretamente relacionados com o uso de técnicas de otimização de modelos de *Machine Learning*.

3.1 Detecção de Anomalias

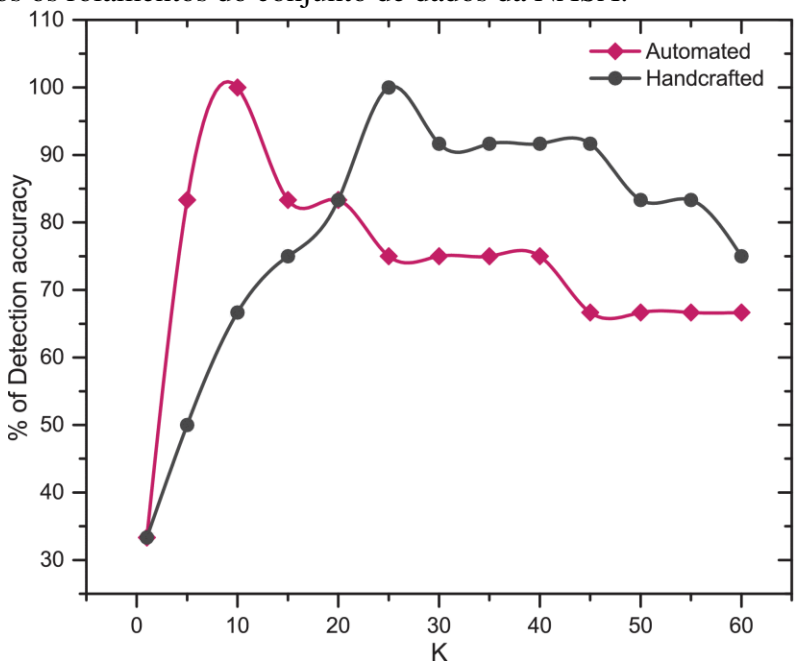
Em ROY *et al.* (2018) é proposto um método automatizado de extração de recursos para monitoramento online de condições de anomalia em rolamentos (J. LEE H. QIU; SERVICES, 2007) baseado em redes neurais do tipo do *autoencoder*, comparando com a extração de características de modo artesanal, representados por cinco recursos extraídos dos sinais brutos. Foram utilizados cinco métodos convencionais de domínio do tempo para criar os recursos artesanais: RMS, curtose, assimetria, fator de crista e pico a pico. Como são 5 métodos aplicados, aos autores utilizam na camada oculta reduzida do *autoenconder* 5 nós. Outro ponto interessante do trabalho está na forma de definir um *threshold* do erro de reconstrução do sinal de entrada. Foi utilizado um método de limiar adaptativo que é baseado na média e no desvio padrão de todos os desvios de saída no momento do treinamento. Esses desvios são uma medida da quantidade de ruído nos dados. Eles definem o *threshold* como visto em (8):

$$T = K(\mu_t + \sigma_t) \tag{8}$$

Onde μ_t é a média de todas as respostas de saída para as amostras de treinamento

de entrada, quando o rolamento está saudável durante sua vida inicial, σ_t é o desvio padrão de todas essas respostas. Há um hiperpâmetro K que foi calculado para o máximo desempenho de ambas as propostas do trabalho. Seus valores e respectiva acurácia estão na Figura 9.

Figura 9 – Valores de K em comparação com a acurácia na determinação do estado de saúde de todos os rolamentos do conjunto de dados da NASA.



Fonte: (ROY *et al.*, 2018).

Embora os valores K para o método automatizado e o método artesanal sejam diferentes, ambas as abordagens podem prever a saúde de todos os rolamentos com 100% de acurácia. Como o método de extração automatizada de características proposto não requer nenhum conhecimento prévio do conjunto de dados e requer treinamento apenas uma vez para um determinado tipo de conjunto de dados, o método proposto é mais viável para aplicação em diferentes máquinas.

O trabalho apresentado em ANDRADE *et al.* (2021) propõe um processamento de dados baseado em uma abordagem de ML não supervisionado para detectar anomalias em estradas, utilizando para isso um microcontrolador e um acelerômetro embarcado em um veículo. A proposta não requer conhecimento prévio dos dados e demanda baixo esforço computacional. É utilizado um algoritmo de detecção de anomalias para fluxos de dados denominado *Typicality and Eccentricity Data Analytics* (TEDA). TEDA utiliza

os conceitos de tipicidade e excentricidade. A primeira está relacionada à similaridade de uma amostra de dados com o restante do conjunto, com base nas distâncias das amostras. Este último é o oposto da tipicidade ao indicar o quão diferente uma amostra é dos outros dados coletados. Portanto, uma amostra com alta excentricidade indica uma possível anomalia.

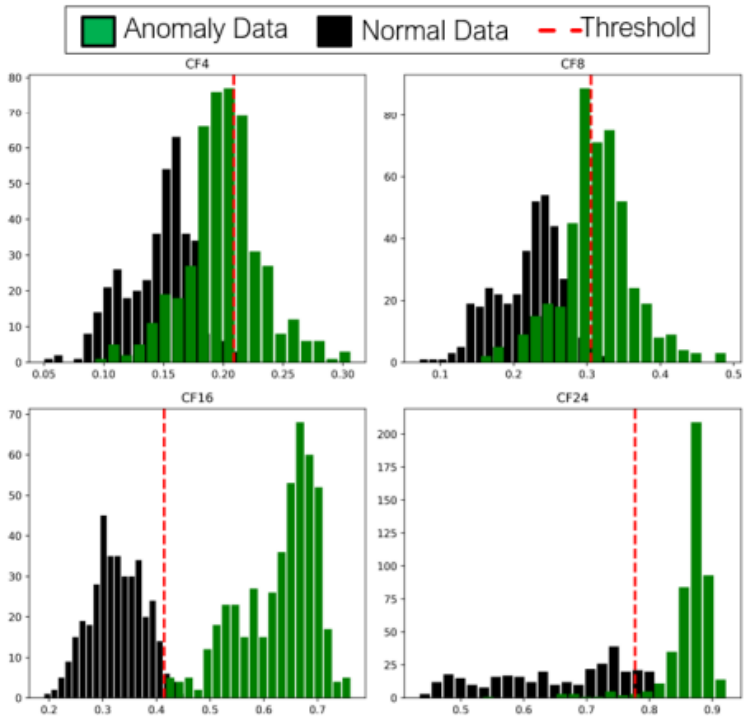
Foram realizados ensaios em um percurso conhecido, com dois diferentes pilotos, dirigindo o mesmo veículo. Os resultados obtidos, segundo os autores, através de experimentos reais foram promissores: o f_1 a média de pontuação foi de 0,76 para o primeiro piloto e 0,78 para o segundo. Isso indica que o modelo classificador alcançou desempenho significativo nos cenários definidos.

Em MOALLEMI *et al.* (2022) os autores apresentam uma implantação completa de um *pipeline* de detecção de anomalias eficiente e escalável para sistemas de monitoramento da saúde estrutural, que não requer o envio de dados brutos a uma central para o processamento, mas depende de computação de borda. Primeiro, são comparadas três abordagens algorítmicas de detecção de anomalias: Análise de Componente Principal (PCA), *Autoencoder* Totalmente Conectado (FC-AE) e *Autoencoder* Convolucional (C-AE). O limite utilizado para distinguir as duas classes de dados, derivou estatisticamente apenas por dados de validação normais. Para calculá-lo, neste caso, é compactado um conjunto de validação de dados normais usando os diferentes algoritmos de compactação. Em seguida, seleciona-se o limiar como a média do MSE sobre todos os dados compactados mais três vezes seu desvio padrão, demonstrado na equação (9).

$$T = (\mu_{MSE} + \sigma_{MSE} \times 3) \tag{9}$$

Os autores exploraram também diferentes fatores de compactação no PCA para analisar seu efeito no desempenho geral da estrutura. Intuitivamente, preservar mais elementos espaciais de alta dimensão não garante melhoria no desempenho geral, pois pode melhorar a reconstrução de dados normais e anormais. Por esta razão, alguns valores para os fatores de compactação foram explorados. Na Figura 10 é mostrado a distribuição dos valores de MSE de anomalias e dados normais com quatro valores de FC. No trabalho, os autores mencionam também as métricas utilizadas para avaliação da solução proposta, todas iguais as descritas na seção 2.7. Comparando o PCA e dois autoencoders diferentes, foi mostrado que o PCA supera os outros dois métodos em aproximadamente 30% e 48%.

Figura 10 – Distribuição dos dados avaliados e a posição do *threshold* utilizada para diferentes fatores de compactação.



Fonte: (MOALLEMI *et al.*, 2022).

3.2 *TinyML*

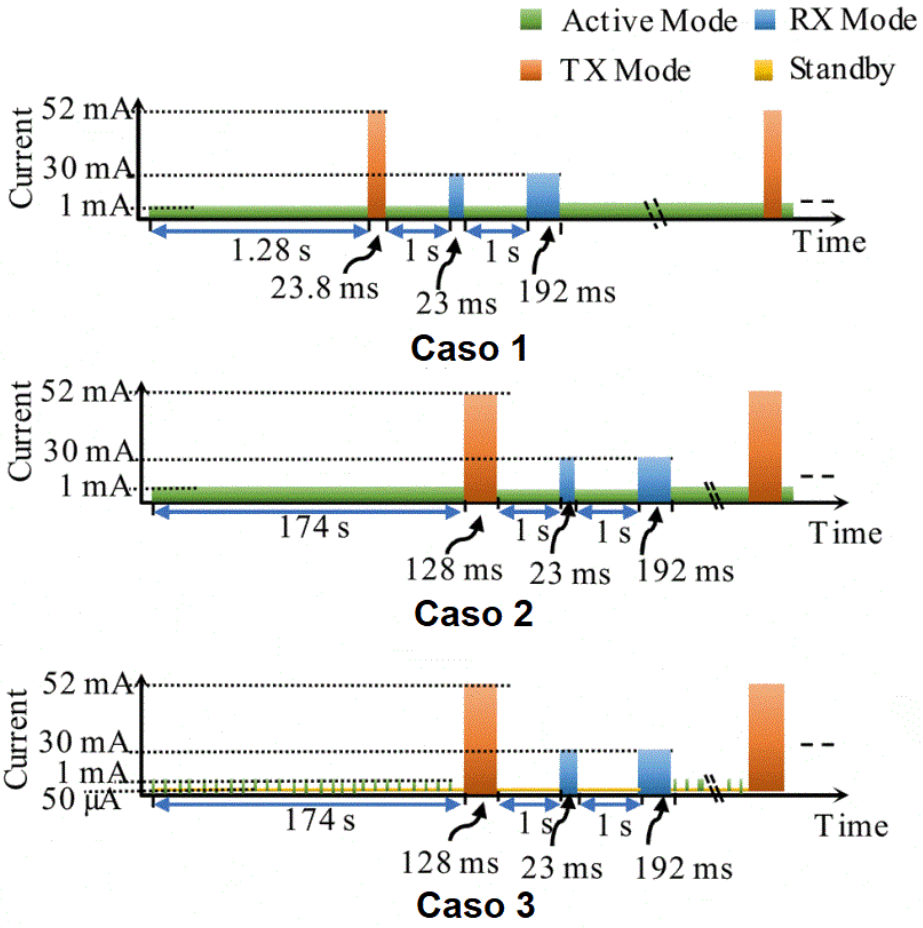
Em SURESH *et al.* (2018) os autores apresentam uma solução que emprega ML em um dispositivo de borda para classificação e realiza transmissão de baixa potência por meio de LoRa. É demonstrado um caso de uso da solução por meio de um teste de campo realizado para classificação de atividade de porcas.

Toda a classificação é realizada pelo algoritmo de *k-Nearest Neighbor* (kNN). O processo de classificação envolve acumular um número de amostras do acelerômetro. As amostras coletadas são então usadas para extrair um conjunto de características comumente usadas, como média, covariância, momentos, etc. O vetor de características extraídas é então alimentado para kNN como um ponto d-dimensional, descobrindo a distância mínima este ponto de um conjunto predefinido de pontos, cada um pertencente a uma classe específica extraída durante a fase de treinamento.

A contribuição do trabalho está em mostrar, por meio de uma comparação em 3 casos distintos, soluções de inferência na borda ou em formato de envio total dos dados a uma

central. Em um primeiro caso, toda classificação realizada no dispositivo de borda é transmitida – emulando uma transmissão total de dados ao *gateway*. No segundo caso, 136 classificações são realizadas na borda e posteriormente enviadas para um *gateway*. O terceiro caso ficou apenas como teórico, onde a única diferença para o segundo é a ativação de um modo *standby* no microcontrolador. Os resultados obtidos estão na Figura 11.

Figura 11 – Casos de operação propostos na analise do trabalho.



Fonte: Adaptado de (SURESH *et al.*, 2018).

Ao implementar o ML incorporado com LoRa, foi possível compactar os dados transmitidos em 512 vezes e prolongar a vida útil da bateria em três vezes. É alcançado um gasto de energia muito baixo de 5,1 mJ por resultado de classificação.

No trabalho já citado anteriormente na seção 3.1, os autores de ANDRADE *et al.* (2021) também aplicaram o conceito de ML em dispositivos com recursos limitados de hardware. Experimentos foram realizados em um cenário real, com a solução final embar-

cada em uma plataforma de hardware Arduino Nano 33. O algoritmo TEDA é executado no microcontrolador, realizando as tarefas de processamento necessárias próximas ao sensor, conforme esperado do paradigma de computação de borda. Os autores não mencionam, ao longo do texto, se houve a utilização de alguma rede sem fio para transmissão a outro ponto, isso foi considerado como um trabalho futuro. A avaliação do estudo de caso levou em conta apenas as métricas de desempenho do algoritmo, não considerando atributos da aplicação, como o consumo de energia do sistema utilizado.

Em MERENDA; PORCARO; IERO (2020) é apresentada uma revisão detalhada sobre modelos, arquitetura e requisitos de soluções que implementam ML de borda em dispositivos de IoT, com o objetivo principal de definir o estado da arte e prever os requisitos de desenvolvimento. Além disso, é fornecido um exemplo de implementação de aprendizado de máquina de borda em um microcontrolador, STMicroelectronics NUCLEO-F746ZG, utilizando um conjunto de dados conhecido, o MNIST (*dataset* com imagens de dígitos manuscritos). Os autores embarcam uma rede neural aplicando as ferramentas disponíveis na biblioteca Tensorflow. A rede neural utilizada no experimento é composta por 784 nós de entrada, duas camadas ocultas de 50 nós e uma camada de saída composta por 10 nós, cada uma representando o dígito a ser classificado. Por fim, é possível proceder à validação para comparar o modelo definido com o gerado em linguagem C pela ferramenta ST, alimentando ambos os modelos com o mesmo conjunto de dados. A validação pode ser realizada como:

1. Validação em desktop: O modelo em C é executado no PC.

2. Validação no embarcado: O modelo gerado é executado no dispositivo de interesse. É necessário carregar o código no microcontrolador e configurar uma comunicação serial para comunicação com o host.

Para ambos os métodos de validação, o resultado da acurácia, segundo os autores, foi de 100% em ambas as avaliações, demonstrando que um modelo de ML pode ser aplicado a uma dispositivo com recursos inferiores do ponto de vista de hardware e, mesmo assim, ter resultados promissores.

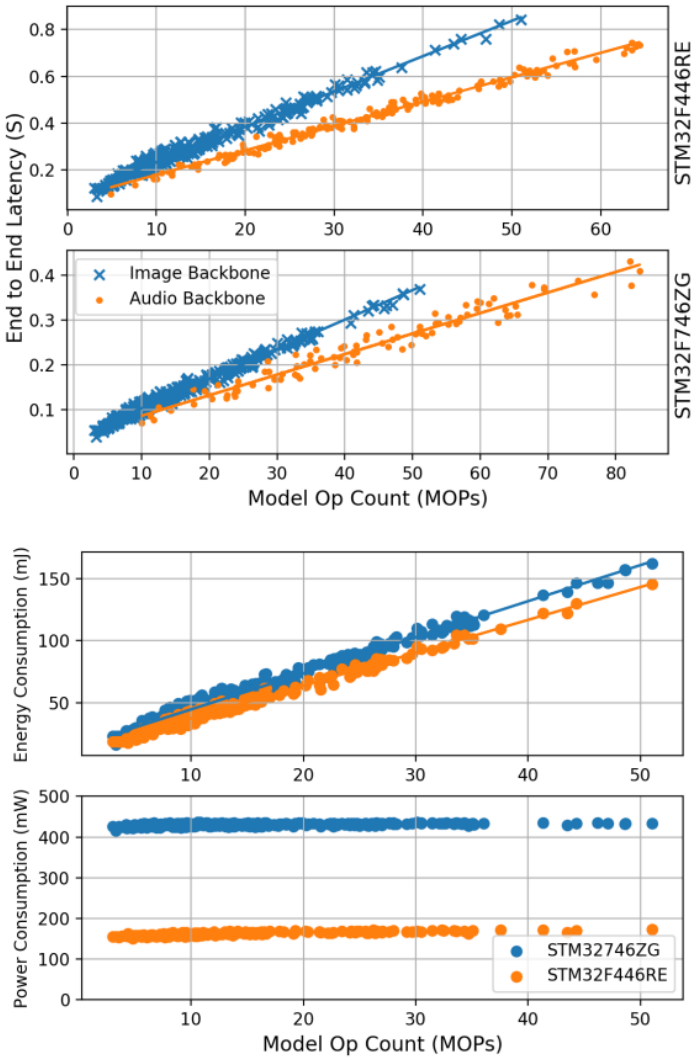
3.3 Otimização dos Modelos de ML

O trabalho descrito em DE PRADO *et al.* (2021) traz uma proposta para veículos de navegação autônoma baseado em dispositivos de baixo recurso. Por se tratar de um sistema que executa inferências baseadas em imagens de uma câmera, os autores utilizam técnicas de *Convolutional Neural Network* (CNN). É denominado de tinyCNN a CNN utilizada para realizar as inferências no dispositivo de borda. A solução foi embarcada em um GAP8, um SoC RISC-V paralelo de ultrabaixa potência, para atender aos requisitos de inferência. O treinamento de uma CNN é normalmente realizado usando dados de ponto flutuante, onde os autores optaram por quantizar os seus modelos. A quantização é um método de compactação que reduz o custo de armazenamento de uma variável empregando precisão numérica reduzida. Além disso, tipos de dados de baixa precisão podem melhorar a intensidade aritmética da CNN, aproveitando o paralelismo em nível de instrução. Assim, foram quantizados os modelos em 8 bits para reduzir a memória de armazenamento e o consumo de energia. Foi empregada a quantização pós-treinamento onde os pesos podem ser quantizados diretamente enquanto as ativações requerem um conjunto de validação para determinar sua faixa dinâmica. Assim, foi observada uma baixa perda de acurácia ($< 3\%$) nos *dataset* iniciais e uma perda desprezível ($< 1\%$) após as fases de aprendizado.

Em BANBURY *et al.* (2020), os autores fazem uma avaliação bastante robusta do *TinyML*. Os resultados experimentais validam a metodologia, gerando modelos chamados de *MicroNet*, que foram implementados em MCUs usando o *Tensorflow Lite Micro*. Eles avaliam latência, consumo de energia e desempenho em diferentes contextos, dentre eles, a detecção de anomalias. São testadas também diferentes arquiteturas de rede neural, principalmente as utilizadas em aplicações que envolvem classificação de imagens. Na arquitetura proposta, os autores alcançaram os melhores resultados dentre as arquiteturas comparadas. Toda a implementação foi realizada em processadores Cortex-M4 e Cortex-M7. Uma contribuição importante do trabalho está na comparação desses dois processadores, do ponto de vista de latência, energia e potência. Na Figura 12, são apresentados alguns resultados gerados. No trabalho, define-se OPS como qualquer operação de multiplicação ou acumulação em uma RNA. Os autores utilizam, para medir a latência de inferência, dois *datasets* distintos e geram modelos com quantidades aleatórias de camadas e pesos. Nota-se que há um crescimento próximo ao linear da latência com o

aumento de operações realizadas, independentemente dos dados ou processador utilizado.

Figura 12 – Medidas de latência, energia e potência em dois diferentes microprocessadores, variando a quantidade de operações realizadas.



Fonte: (BANBURY *et al.*, 2020).

Nos gráficos que reportam a energia e potência, os autores consideram que para um determinado MCU, a energia é independente do tamanho e do design do modelo da rede neural. Portanto, a energia por inferência é uma função do tamanho do MCU, que determina a potência, e do número de operações, que determina a latência.

Por fim, o trabalho realizado em HAN; MAO; DALLY (2016) apresenta a compressão de modelos de aprendizado de máquina. Os autores abordam como as redes neurais são intensivas em computação e memória, tornando-as difíceis de implantar em sistemas embarcados com recursos de hardware limitados. Eles abordam 3 métodos de compres-

são: *pruning*, quantização pós-treinamento e agrupamento de pesos. A poda de pesos reduz o número de conexões de 9 a 13 vezes; A quantização então reduz o número de bits que representam cada conexão de 32 para 5. No conjunto de dados de objetos visuais, o método reduziu o armazenamento exigido em 35x, de 240 MB para 6,9 MB, sem perda de precisão. O método reduziu o tamanho do VGG-16 – padrão de rede utilizado para classificação de imagens – em 49x: de 552 MB para 11,3 MB, novamente sem perda de precisão. Entretanto, o trabalho em questão não faz uma abordagem voltada para dispositivos com baixos recursos, definidos na seção 2.4.

3.4 Resumo

A Tabela 1 apresenta uma visão geral dos principais trabalhos avaliados. Nela são apresentadas as diferenças mais relevantes entre os trabalhos propostos na literatura, assim como a proposta desenvolvida neste trabalho. Cabe salientar, que mesmo diante de todo o esforço de pesquisa bibliográfica, não foram encontrados trabalhos como o proposto, principalmente por aplicar técnicas de otimização dos modelos de *TinyML* e realizar uma avaliação em uma arquitetura de LPWAN.

Tabela 1 – Relação de trabalhos e suas contribuições.

	Avaliação de Aplicação		*Edge Computing*	Otimização de Modelos de ML			
	Consumo de Energia	Detecção de Anomalia	*TinyML* em LPWAN	Quantização Pós-Treino	Treinamento Consciente de Quantização	*Pruning*	Agrupamento de Pesos
(ROY *et al.*, 2018)	-	x	-	-	-	-	-
(SURESH *et al.*, 2018)	x	-	x	-	-	-	-
(MOALLEMI *et al.*, 2022)	x	x	x	-	-	-	-
(ANDRADE *et al.*, 2021)	-	x	x	-	-	-	-
(MERENDA; PORCARO; IERO, 2020)	-	-	-	-	-	-	-
(BANBURY *et al.*, 2020)	x	-	-	x	x	-	-
(HAN; MAO; DALLY, 2016)	-	-	-	x	-	x	x
(DE PRADO *et al.*, 2021)	x	-	-	x	-	-	-
Este Trabalho	x	x	x	x	x	x	x

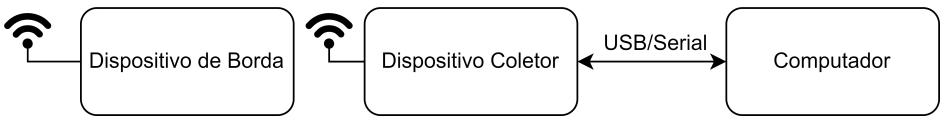
4 MÉTODOS E MATERIAIS

Neste capítulo são apresentados os métodos e materiais necessários para o desenvolvimento das avaliações das técnicas de otimização em modelos de ML. Além disso, são apresentadas as ferramentas empregadas para a realização dos estudos de caso em uma LPWAN, com a utilização do protocolo LoRa. O hardware usado é apresentado na seção 4.1 e,o software adotado em diferentes aspectos da solução, na seção 4.2. A seção 4.3 apresenta as definições do circuito de instrumentação para medir o consumo de energia. Na seção 4.4, são descritos os *datasets* utilizados.

4.1 Hardware

A topologia utilizada no *setup* de testes é apresentada na Figura 13. O setup é composto por um PC, um coletor e o dispositivo de borda. Nas seguintes subseções são apresentados os dispositivos que fazem parte da estrutura da rede e as suas principais funções. O computador empregado utiliza como sistema operacional o Ubuntu 20.04.2.

Figura 13 – Arquitetura básica em termos de hardware utilizado.



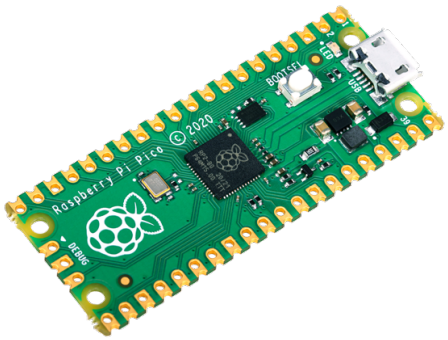
Fonte: Autor.

4.1.1 Raspberry Pi Pico

Para o desenvolvimento dos estudos de caso, aplicou-se o uso de uma placa de desenvolvimento Raspberry Pi Pico, presente na Figura 14. O dispositivo conta com dois núcleos ARM Cortex-M0+ com um clock de até 133 MHz. Além disso, a placa possui 256

KB de SRAM, 2 MB de memória Flash, sensor de temperatura *on-board*, um *Real Time Counter* (RTC) entre outros periféricos. A escolha deve-se ao baixo custo do dispositivo, um SDK (*Software Development Kit*) robusto, e a compatibilidade com a ferramenta de conversão de modelos TensorFlow, o *TensorFlow Lite Micro*.

Figura 14 – Placa de desenvolvimento utilizada no desenvolvido do trabalho.



Fonte: (PI, 2022)

A placa citada anteriormente exerce o papel de dispositivo de borda na arquitetura que foi apresentada na Figura 13.

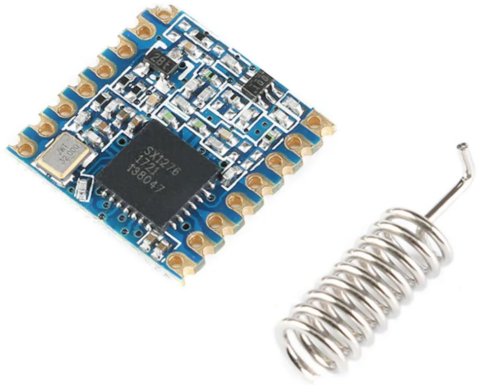
4.1.2 Comunicação LoRa

Considerando a proposta de se ter uma aplicação que esteja inserida no contexto de uma LPWAN, foi adotada a faixa de operação na faixa de 915 MHz. O protocolo de comunicação *Long Range*, o popular LoRa, desenvolvido pela empresa Cycleo e adquirida pela Semtech, consegue unir baixo consumo de energia com grande alcance de comunicação (SEMTECH, 2022).

Para o segmento do trabalho, utilizou-se o transceptor LoRa RFM95W (Figura 15), o qual utiliza o circuito integrado (CI) controlador SX1276 e um módulo com faixa de transmissão entre 868 e 915 MHz, com taxa de dados de até 37,5 kbps e sensibilidade de enlace entre -111 e -148 dBm, conforme a largura de banda utilizada (SEMTECH, 2022).

O LoRa é baseado na tecnologia *Chirp Spread Spectrum* (CSS), onde os símbolos (*chirps*) são os portadores de dados. O fator de espalhamento controla a taxa de *chirp* e, portanto, controla a taxa de transmissão de dados. Fatores de espalhamento mais baixos significam *chirps* mais rápidos e, portanto, uma taxa de transmissão de dados mais alta.

Figura 15 – Módulo LoRa.



Fonte: Autor.

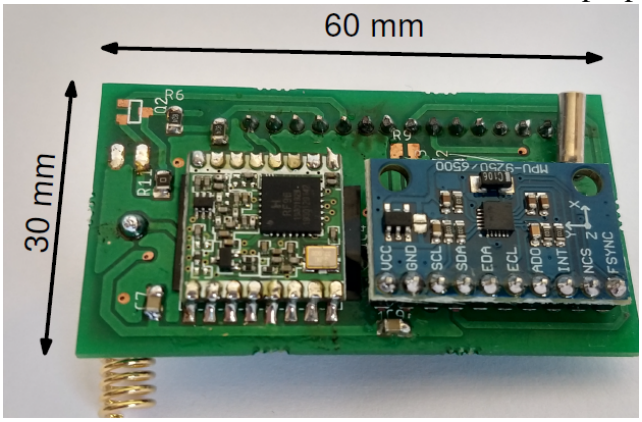
Para cada aumento no fator de espalhamento, a taxa de varredura de *chirp* é reduzida pela metade e, portanto, a taxa de transmissão de dados é reduzida pela metade. Fatores de espalhamento mais baixos reduzem o alcance das transmissões LoRa, pois reduzem o ganho de processamento e aumentam a taxa de bits. A alteração do fator de espalhamento permite que a rede aumente ou diminua a taxa de dados para cada dispositivo final ao custo do alcance.

O transceptor está presente em dois pontos da arquitetura de hardware vista na Figura 13. Ele está conectado ao dispositivo de borda, visto na subseção 4.1.1, via *Serial Peripheral Interface* (SPI), para coletar as informações e enviar até o ponto de acesso, que será descrito a seguir.

4.1.3 Módulo Coletor

O coletor de dados que faz o papel de gateway da solução proposta foi desenvolvido em VALADÃO *et al.* (2021). O autor utilizou o mesmo módulo LoRa descrito na seção 4.1.2. O dispositivo conta com um microcontrolador ATmega328P. Este microcontrolador opera com clock de 20MHz e é capaz de executar 20 milhões de instruções por segundo. O dispositivo faz a interface entre o hardware de borda com as aplicações que estão sendo executadas em um computador.

Figura 16 – Coletor de dados utilizado no sistema proposto.



Fonte: Adaptado de (VALADÃO *et al.*, 2021).

4.2 Software

Uma parte importante da implementação deste trabalho está nos recursos de software. As próximas subseções tratam de diferentes momentos da construção dos modelos de ML e do processo portabilidade desses modelos para a solução de hardware descrita na seção 4.1.

4.2.1 *TensorFlow*

TensorFlow é uma interface para execução de algoritmos de aprendizado de máquina e sua implementação para compilação desses algoritmos. Uma computação expressa usando o *TensorFlow* pode ser executada com pouca ou nenhuma alteração em uma ampla variedade de sistemas heterogêneos, desde de dispositivos móveis como telefones e tablets até sistemas distribuídos de larga escala com centenas de máquinas e milhares de dispositivos. O sistema é flexível e pode ser usado para expressar uma ampla variedade de algoritmos, incluindo os de treinamento e inferência para modelos de redes neurais profundas, e tem sido utilizado para condução de pesquisa e para implementação de sistemas de aprendizado de máquina (ABADI *et al.*, 2015). O *TensorFlow* possibilita a criação e execução uma rede do tipo *autoencoder*, descrita na subseção 2.6.1.

4.2.2 *TensorFlow Lite*

Grandes empresas tecnológicas estão promovendo a expansão do ecossistema TinyML, oferecendo bibliotecas de desenvolvimento de código aberto. O Google lançou o *Tensor-*

Flow Lite (LITE, 2019), que inclui um conjunto de ferramentas para adaptar os modelos do *TensorFlow* com o objetivo de torná-los executáveis em dispositivos móveis e embarcados. Esse *framework* é composto por dois elementos principais, a saber, o conversor, que porta os modelos do *TensorFlow* para código otimizado que pode ser executado em plataformas restritas, e o interpretador, que realmente executa o código gerado pelo conversor.

4.2.3 *Software Development Kit*

O *Software Development Kit* (SDK) fornece os *headers*, bibliotecas e sistema de compilação necessários para escrever programas para dispositivos baseados em RP2040, como Raspberry Pi Pico em linguagem assembly C ou C++ (LTD, 2022). Toda a aplicação embarcada no dispositivo de borda, que realiza a leitura dos dados, faz a inferência e envia pacotes com informações foi desenvolvida sobre o SDK. Complementando a arquitetura de software, são necessárias ferramentas de integração com LoRa (ALLAN, 2021). Todo o *stack* de configuração das características que a rede LoRa terá, são realizadas na inicialização da aplicação.

4.3 Consumo de Energia

Neste trabalho, em um dos estudo de caso, será avaliado o consumo de energia em dois modos de operação propostos. Como se está trabalhando com dispositivos de baixa potência, na ordem de miliwatts, a aferição de tensão e corrente necessita da construção de um circuito de instrumentação para amplificar o sinal de interesse.

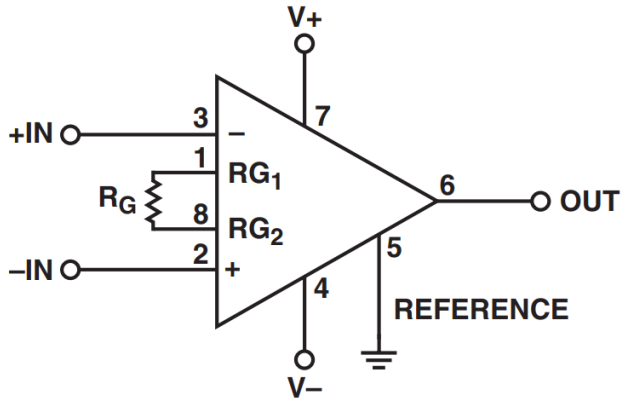
4.3.1 Amplificador de Instrumentação

O AMP02 é o primeiro amplificador de instrumentação de precisão disponível em um *footprint* de 8 pinos, conforme ilustrado na Figura 17. O ganho do AMP02 é definido por um único resistor externo e pode variar de 1 a 10000, conforme equação (10) (ANALOG DEVICES, 2022).

$$G = \frac{50k\Omega}{R_G} + 1 \tag{10}$$

Nenhum resistor de ajuste de ganho é necessário para ganho unitário. O AMP02 inclui uma rede de proteção de entrada que permite que as entradas sejam levadas 60 V

Figura 17 – Conexões básicas do amplificado de instrumentação de precisão.



Fonte: (ANALOG DEVICES, 2022).

além de qualquer barramento de alimentação sem danificar o dispositivo. Algumas das aplicações típicas para o uso desse componente são: Amplificador Diferencial, Amplificador *Strain Gage*, Amplificador de termopar, Amplificador de Instrumentação de Ganho Programável, Sistemas de Aquisição de Dados, entre outros.

4.3.2 Circuito de avaliação de corrente

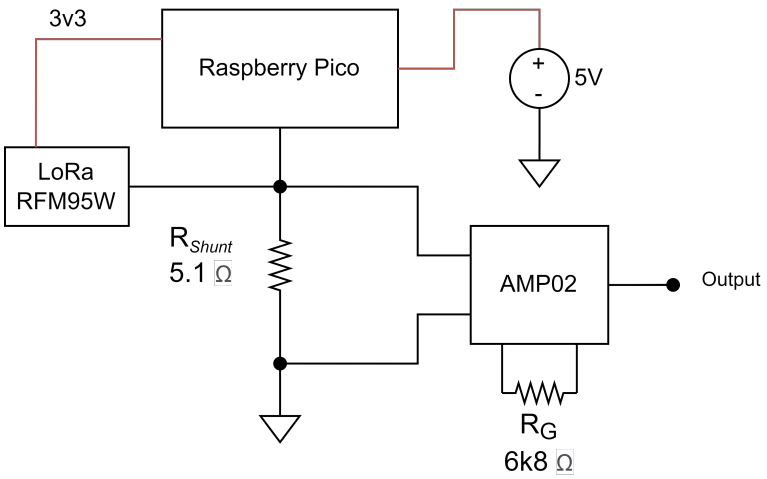
Para avaliar o consumo de energia do dispositivo de borda, foi desenvolvido uma circuito para medir a tensão sobre um resistor de baixo valor, de valor conhecido, comumente chamado de *shunt* (R_{shunt}). As conexões relevantes deste circuito são mostradas na Figura 18.

O dispositivo RFM95W é alimentado pela placa de desenvolvimento Raspberry Pi Pico. O AMP02 é conectado a uma fonte simétrica de +5V e -5V. Desejando amplificar o sinal de entrada em aproximadamente 8 vezes, valor de R_G é calculado pela equação 10, resultando em:

$$R_G = \frac{50k\Omega}{8-1} = 7,14k\Omega \approx 6,8k\Omega \tag{11}$$

É feita uma aproximação para um resistor de valor comercial típico. A amplificação deste sinal foi realizada com base na descrição feita em (SURESH *et al.*, 2018), onde a partir dos resultados das medições, foi descoberto que o módulo consome 52 mA, 30 mA e 1 mA de corrente nos estados de transmissão, captura de dados e processamento (inferência), respectivamente. Essa baixa corrente no momento da inferência pode ser observada na Figura 19.

Figura 18 – Circuito simplificado de avaliação de corrente.



Fonte: Autor.

O equipamento utilizado para a análise dos sinais de tensão foi o osciloscópio Agilent MSO9104A, o qual possui filtros digitais de sinais. Na avaliação, foi utilizado um filtro passa baixa com frequência de corte em 10 kHz, removendo assim os ruídos de alta frequência. Outros dois equipamento utilizados na implementação são: uma fonte Minipa MPL-3305M e um multímetro Fluke 115. A fonte possui dois canais distintos, que são ligados em paralelo para servirem de fonte simétrica de tensão +5V e -5V, que alimenta o AMP02 em seus terminais V- e V+ (Figura 17). O multímetro é utilizado para verificar o valor do resistor *shunt*, haja vista a tolerância do componente de $\pm$ 10%.

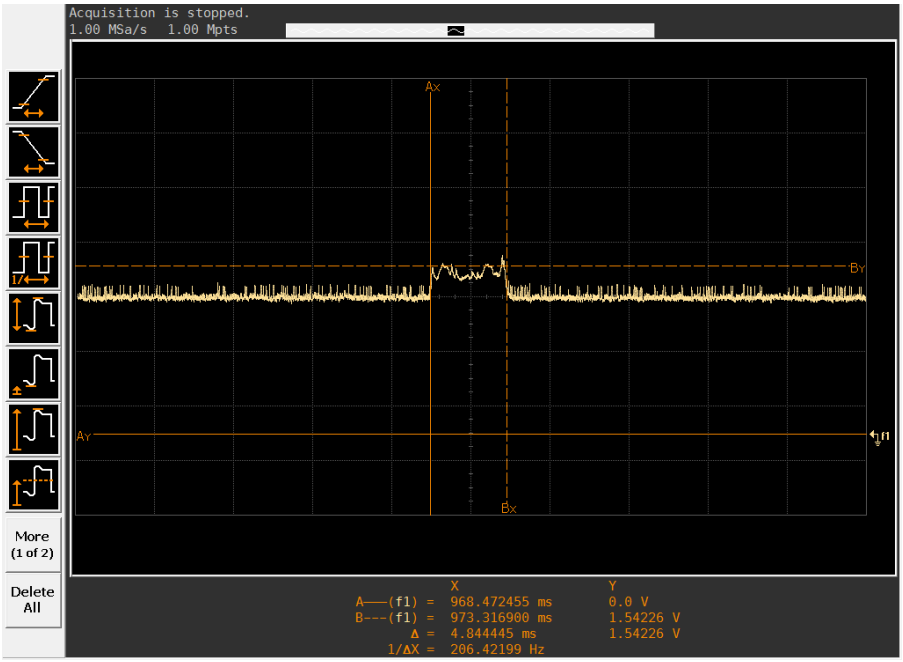
4.4 Conjunto de dados

Na elaboração de um sistema de detecção de anomalias, com a inferência sendo realizada em dispositivos de borda, existe a necessidade de se ter uma fonte de dados. Para tanto, duas alternativas são abordadas: a primeira delas é a geração de dados através de uma biblioteca escrita em *python*; a segunda opção aplicada é a utilização de um *dataset* real.

4.4.1 *Dataset - ADBench*

No *Anomaly Detection Benchmark* (ADBench) (HAN *et al.*, 2022), é possível criar conjuntos de dados sintéticos realistas a partir de conjuntos de dados do mundo real inje-

Figura 19 – Visualização aproximada do momento da inferência, em termos de tensão sobre o resistor *shunt*.



Fonte: Autor.

tando tipos específicos de anomalias. A biblioteca desenvolvida, PyOD, permite a criação de conjunto de dados customizável. Os dados normais são gerados por uma distribuição gaussiana multivariada e os *outliers* são gerados por uma distribuição uniforme. O trecho de código apresentado na sequência descreve os parâmetros que devem ser fornecidos, de acordo com o que se deseja gerar. Os parâmetros n_train e n_test indicam a quantidade de amostras para treinamento e teste, respectivamente.

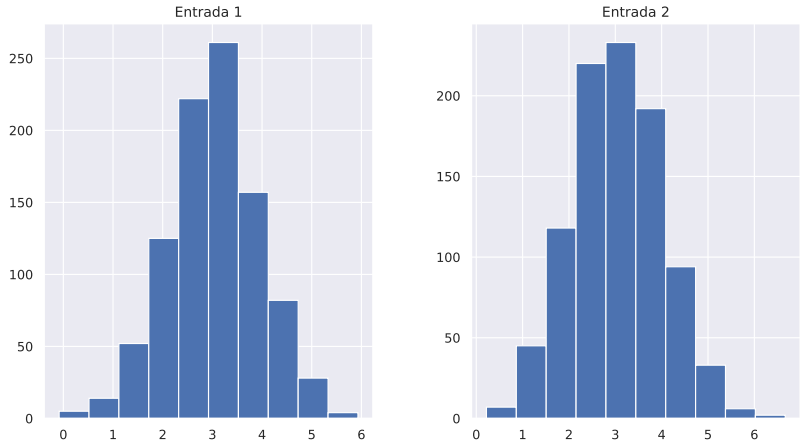
```
from pyod.utils.data import generate_data
X_train, X_test, y_train, y_test = \
generate_data(n_train=1000,
              n_test=1000,
              n_features=2,
              contamination=0.1,
              offset = 10)
```

Já $n\_features$, indica quantos serão os dados de entrada. No $contamination$ é passado o percentual, em formato decimal, de amostras que serão anormais no *dataset*.

Por fim, o último parâmetro, $offset$, ajusta o intervalo de valores da distribuição gaussiana para os valores normais e da uniforme para os *outliers*. Esta variação pode ser

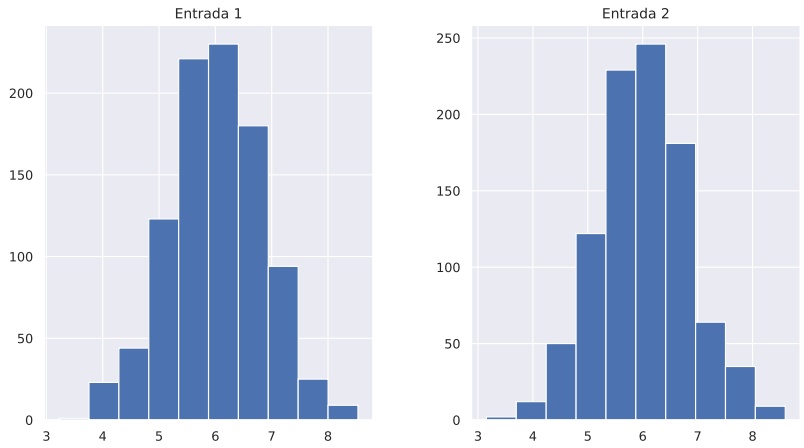
observada nos histogramas das figuras 20 e 21, onde os valores das amostras de treinamento, com de $offset$ 5 e 10 respectivamente, são apresentadas.

Figura 20 – Histograma de amostras normais de treinamento com $offset$ = 5.



Fonte: Autor.

Figura 21 – Histograma de amostras normais de treinamento com $offset$ = 10.

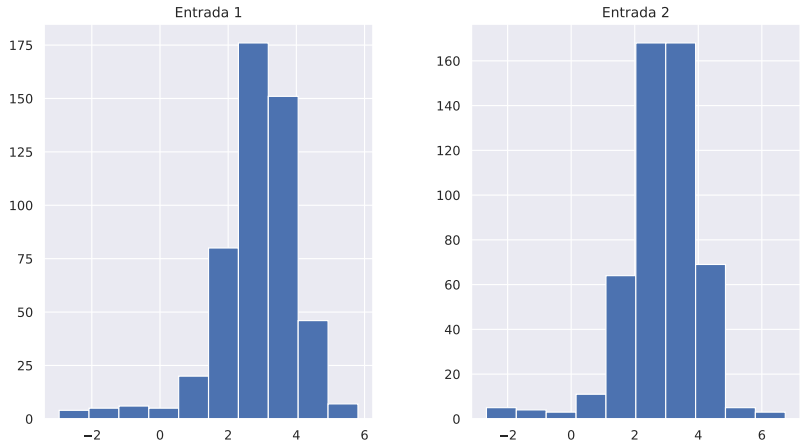


Fonte: Autor.

Essa variação nos valores da distribuição faz com que as amostras geradas como anomalias fiquem mais próximas do valor médio das amostras normais, o que irá implicar em maior dificuldade de se distinguir uma da outra. Em ambas as figuras, apenas os dados normais de treinamento estão contemplados. Quando visualizado o histograma do conjunto de teste, onde as amostras rotuladas como anômalas estão presentes, a distribuição

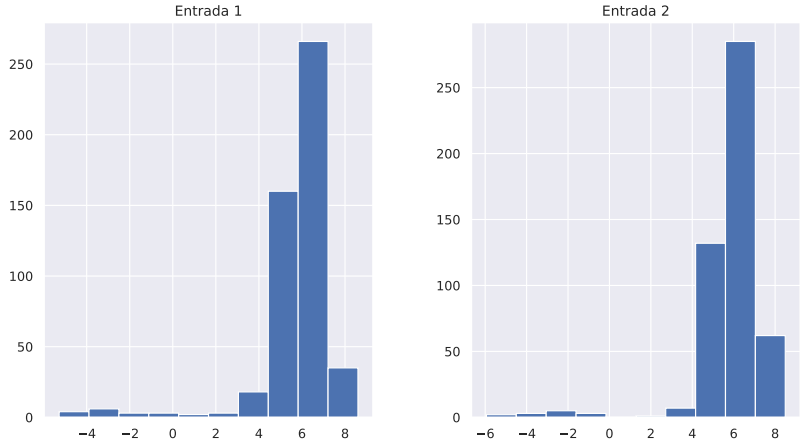
fica representada pelas figuras 22 e 23.

Figura 22 – Histograma de amostras de teste com $offset$ = 5.



Fonte: Autor.

Figura 23 – Histograma de amostras de teste com $offset$ = 10.



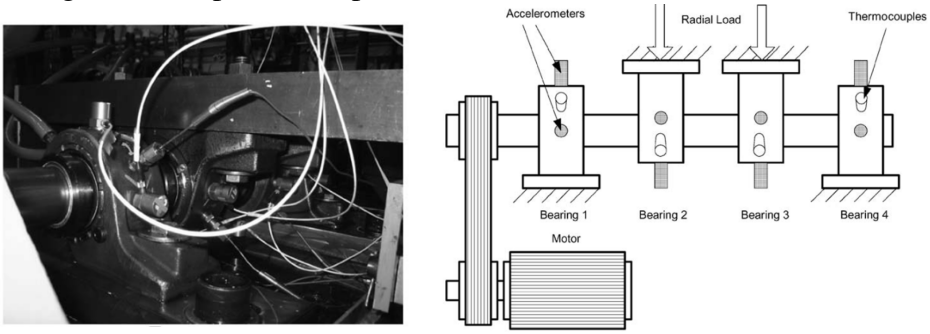
Fonte: Autor.

É clara a diferença causada pela redução da distribuição gaussiana, deixando os valores contaminados mais distantes da média, o que pode mudar completamente o resultado do desempenho do modelo de ML avaliado neste trabalho.

4.4.2 *Dataset* NASA

Neste trabalho, é utilizado o conjunto de dados NASA *Bearing* (J. LEE H. QIU; SERVICES, 2007), fornecido pelo *Center for Intelligent Maintenance Systems* (IMS), da Universidade de Cincinnati. Existem três tipos de conjuntos de dados, cada um deles consistindo em sinais de vibração de quatro rolamentos diferentes, conforme ilustra a Figura 24.

Figura 24 – Aparelho Experimental (Anel de Teste de Rolamento).



Fonte: Adaptado de (J. LEE H. QIU; SERVICES, 2007).

O conjunto de dados 1 contém sinais de vibração para todos os quatro rolamentos que foram adquiridos pelos acelerômetros para os eixos X e Y. Para os conjuntos de dados 2 e 3, os sinais de vibração para todos os quatro rolamentos foram obtidos apenas dos acelerômetros ao longo do eixo X. Esses sinais de vibração foram registrados durante 1 segundo, em um intervalo de 10 minutos, com uma taxa de amostragem de 20 KHz. Uma sumarização das informações desse conjunto de dados está na Tabela 2.

Tabela 2 – Descrição sumarizada dos *datasets* de monitoramento de rolamentos coletados pela NASA.

Setup	Data Início	Data da Falha	Falha	Números de Amostras
Dataset 1	22/10/2003 12:06:04	25/11/2003 23:39:56	Rolamento 3 e 4	2156
Dataset 2	12/02/2004 10:32:39	19/02/2004 06:22:39	Rolamento 1	984
Dataset 3	04/03/2004 09:27:46	04/04/2004 19:01:57	Rolamento 3	6324

5 IMPLEMENTAÇÃO

Neste capítulo são apresentadas as questões relativas à implementação de um sistema para avaliação das técnicas de otimização de modelos de ML. O primeiro aspecto é a definição da arquitetura do sistema proposto para avaliar as técnicas de otimização. Em 5.1, são definidos e explicados os passos para se criar uma solução de *TinyML* e avaliação de um cenário de aplicação em uma LPWAN.

Por fim, é apresentada na seção 5.2 a avaliação do desempenho de cada técnica de otimização abordada. Para tal, são colocadas também as características da RNA utilizada, definições no modo de treinamento do modelo e as avaliações desse treinamento.

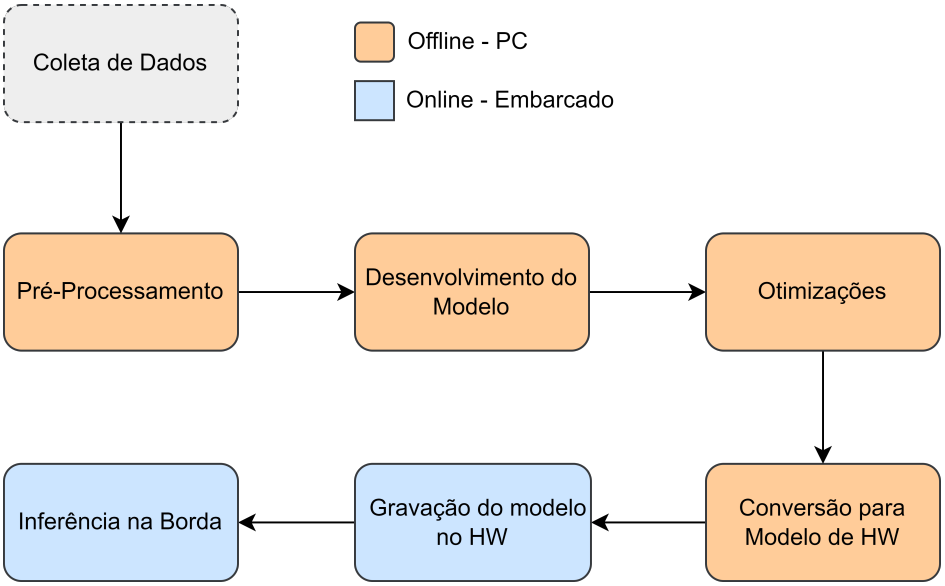
5.1 Arquitetura e sua avaliação

Para realizar a avaliação dos modelos otimizados e apresentar os benefícios de se realizar inferências em dispositivos de borda, é necessário estabelecer uma arquitetura. Com muitas etapas, como visto na Figura 25, alguns critérios devem ser estabelecidos. O primeiro conceito a ser definido é onde o processo está sendo executado. Os blocos marcados como *Offline - PC* são executados em um computador de forma *Offline* - fora da operação em tempo real no qual tem-se o conhecimento de todo o conjunto de treinamento ou da resposta esperada da rede (PARMA; MENEZES; BRAGA, 1999).

Todos os processos são realizados em linguagem Python. Como pilar central do desenvolvimento, são exploradas as funcionalidades e ferramentas disponíveis no *framework* TensorFlow (ABADI *et al.*, 2015).

1. Pré-processamento: análise de um conjunto de dados, onde são observadas as características principais, normalização e padronização dos dados. Como a proposta está centrada em detecção de anomalias, é neste momento que são separadas as

Figura 25 – Etapas da construção de uma solução *TinyML*.



Fonte: Autor.

amostras que possuem o comportamento esperado (normal) da operação e as amostras que apresentam a anomalia. É nesse momento também que o conjunto de dados é dividido em amostras de treinamento e teste.

2. Desenvolvimento do modelo: dentro do contexto de detecção de anomalias, muitos são os algoritmos que podem ser abordados. Nesse trabalho, foi aplicado o algoritmo de autoencoder.

3. Otimizações: todas as otimizações descritas seção 2.5 são aplicadas neste momento.

4. Conversão para Modelo de Hardware (HW): por fim, para cada modelo otimizado, um arquivo é gerado. Posteriormente, cada arquivo é anexado ao software embarcado, gerando um binário por otimização.

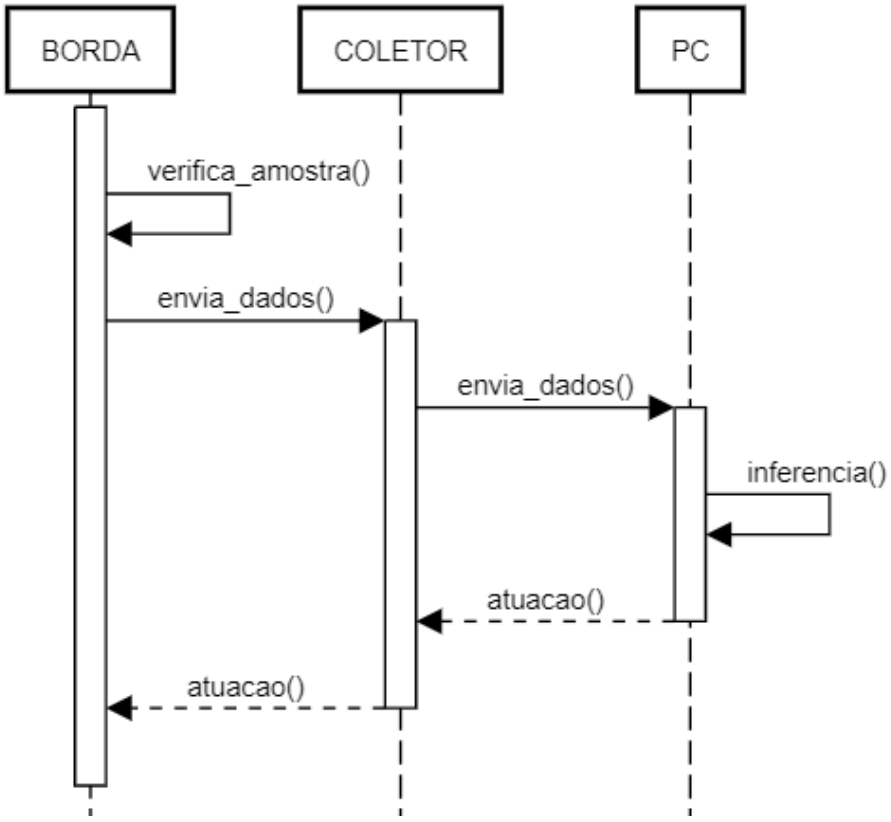
Os blocos da Figura 25 em azul, legendados como *Online - Embarcado*, são tratados no dispositivo de borda. O hardware adotado para este trabalho é uma Raspberry Pi Pico, descrito na seção 4.1.1.

Estabelecido o fluxo dos processos necessários para a construção de uma solução *TinyML*, faz-se necessária a criação de um cenário onde se possa aplicar a inferência na borda. Assim, é possível verificar os benefícios da arquitetura da solução. O formato da aplicação é a de detecção de anomalias.

Para a criação dessa aplicação, uma LPWAN do tipo *LoRa* é utilizada. Uma rede é estabelecida entre o dispositivo da borda e um dispositivo coletor de informações - um Arduino Nano, neste caso. Ambos os dispositivos são dotados de um módulo *LoRa*. O módulo SX1276 é um transceptor, dispositivo responsável pela comunicação sem fio, oferecendo comunicação de espectro disperso de alcance ultra longo e alta imunidade a interferências e minimizando o consumo de energia. É utilizada a camada física do protocolo *LoRaWAN*. O fator de espalhamento aplicado é o de coeficiente 8 e a frequência de operação utiliza a banda de 915 MHz.

Para a avaliação desse cenário, são elaborados dois modos de operação. O primeiro trata de quando o dispositivo de borda envia o conteúdo completo da amostra, ou seja, faz uma transferência total do conteúdo de dados disponível, como mostra o diagrama de sequência na Figura 26. O dispositivo coletor está conectado a um computador por uma interface USB-Serial, que realiza a inferência do dado.

Figura 26 – Diagrama de sequência adotado para a avaliação da arquitetura proposta, no modo de operação *cloud*.

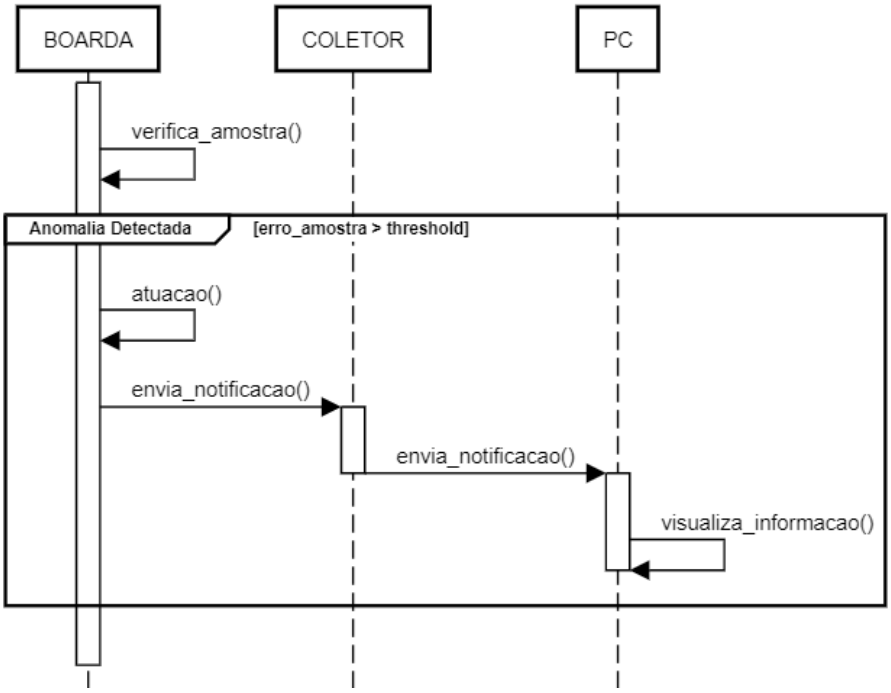


Fonte: Autor.

Por conta deste processo, qualquer ação de atuação após a inferência no PC acarretaria em um retorno para o dispositivo de borda. Este trabalho não aborda a atuação haja vista o teor da proposta, que se concentra em avaliar métodos de otimização de modelos de ML.

O segundo modo de operação, ilustrado na Figura 27, realiza a inferência no dispositivo de borda e faz um envio ao coletor apenas quando existir algum tipo de anomalia. Nesse modo de operação, não são enviados os dados de entrada, apenas é informado qual o sensor que apresentou o maior erro na reconstrução do sinal de entrada. Notar que existe apenas uma direção na comunicação, visto que todo o processo de decisão e atuação está na borda do sistema.

Figura 27 – Diagrama de sequência adotado para a avaliação da arquitetura proposta, no modo de operação *edge*.

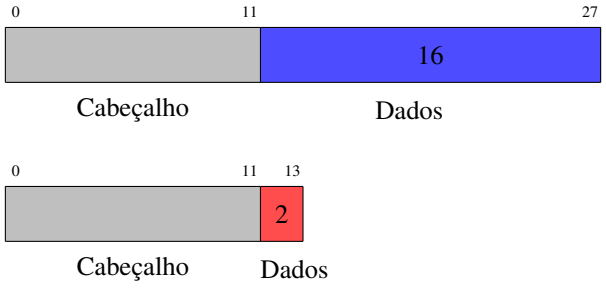


Fonte: Autor.

Os fatores considerados preponderantes para comparação dos dois modos de operação serão o consumo de energia e a diferença de tempo para se ter a inferência realizada. A forma de se medir o consumo realizado foi descrita na seção 4.3.2. Para realizar aferição do tempo de inferência na borda, é utilizado o contador interno do microcontrolador RP2040, que apresenta uma resolução de 1 μs. Para o modelo da Figura 26, são contabilizados os tempos de transmissão pela rede LoRa.

Em ambos os modos de operação, o mesmo cabeçalho de 12 bytes é anexado. O ID do dispositivo, data/hora e um espaço para implementação de comandos são anexados ao *payload*. Quando não há inferência na borda, é necessário enviar todos os dados amostrados – 16 bytes por amostra, totalizando 28 bytes por pacote enviado. Já para a inferência realizada no hardware de borda, a carga útil é de apenas dois bytes, utilizados para a demanda da aplicação (informar o rolamento com problema). A concatenação dos pacotes é ilustrada na Figura 28. Em cinza está o cabeçalho (*header*), igual para ambos os sistemas, em azul a carga útil no modo de operação *cloud* e em vermelho o modo de inferência na borda.

Figura 28 – Estrutura de um pacote enviado em ambos as soluções propostas.



Fonte: Autor.

5.2 Avaliação das Técnicas de Otimização

A metodologia de teste da arquitetura do sistema proposto possui uma etapa não mapeada na Figura 25: a coleta de dados. Nesse trabalho, são adotadas formas diferentes de obtenção de dados. A primeira delas é gerando *datasets* por intermédio do biblioteca apresentada na subseção 4.4. Conforme apresentado, são necessários parâmetros de entrada para a obtenção do *dataset*.

Em um primeiro momento, os elementos para geração dos dados são descritos no trecho de código abaixo. Algo que é levado em consideração para gerar esses dados está na característica de aplicação que se está abordando nesse trabalho. Aplicações que fazem detecção de anomalias trabalham a maior parte do tempo sob condições normais, sem apresentarem eventos adversos. Por isso, a contaminação gerada é de apenas 5% do conjunto de dados.

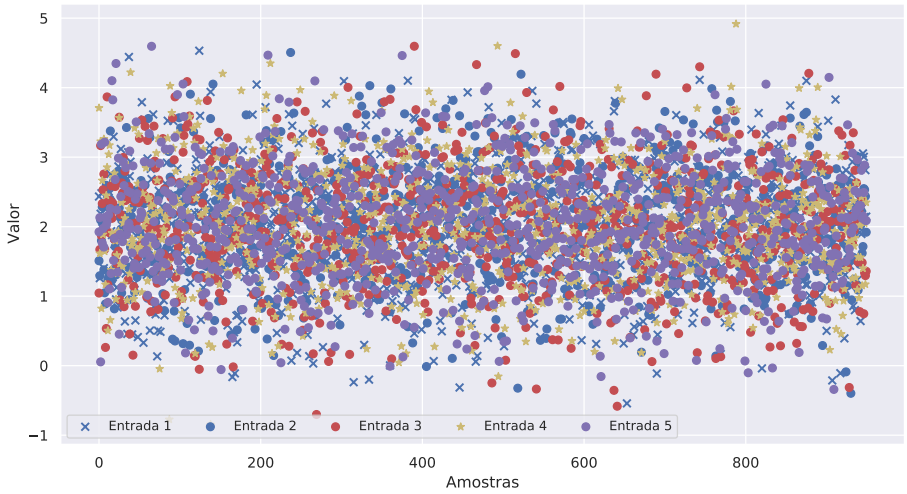
```
%% Trecho de codigo utilizada para gerar datasets %%
from pyod.utils.data import generate_data
n_features_values = [5,10,15,20,25]
offset_values = [5,10]
X_train, X_test, y_train, y_test = \
generate_data(n_train       = 1000,
              n_test        = 500,
              n_features    = n_features_values,
              contamination = 0.05,
              offset        = offset_values)
```

Outro aspecto está na quantidade de amostras de treinamento e teste geradas. Como se está trabalhando com redes do tipo *autoencoder*, que são não-supervisionadas e buscam a simples reconstrução do sinal de entrada com o menor erro possível, não há necessidade de se estabelecer algum padrão ou balanceamento dos dados. A natureza da aplicação de detecção de anomalias tem por característica um desbalanceamento dos dados, pois há maior ocorrência de amostras normais em relação a amostras com anomalia.

Os valores de *offset* utilizados diferem no seu propósito: os conjuntos de dados gerados com o valor 10, são avaliados apenas por tempo de inferência e tamanhos dos modelos gerados após se aplicarem as técnicas de otimização. Quando se estiver avaliando os modelos no ponto de vista de desempenho, ou seja, sua capacidade de determinar uma amostra com ou sem anomalia, o valor utilizado é 5. Essa distinção, usando um valor menor, busca reduzir a distribuição gaussiana dos valores anômalos, deixando-os mais próximos a média e, consequentemente, mais difíceis de serem identificados, conforme apresentado na seção 4.4.

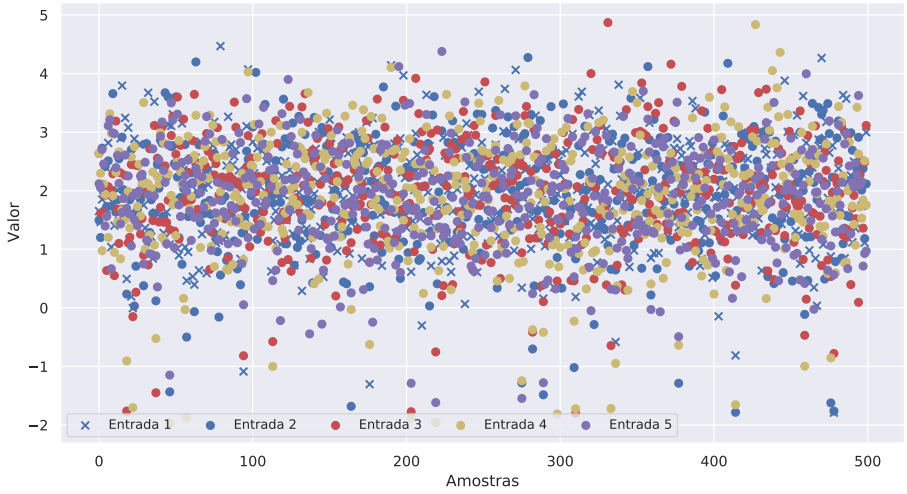
Uma lista de valores ($n\_features\_values$) é criada para a geração dos dados. O que está se buscando nessa implementação é ter 5 *datasets*, variando a quantidade de sinais de entrada da rede neural. Essa diferenciação será importante para avaliação nos estudos de caso que serão expostos na próxima seção. As figuras 29 e 30 apresentam, a distribuição do conjunto de dados que possui 5 entradas e *offset* = 5.

Figura 29 – Conjunto de dados com 5 características de entrada utilizado no treinamento do modelo de ML.



Fonte: Autor.

Figura 30 – Conjunto de dados com 5 características de entrada utilizado no teste do modelo de ML.

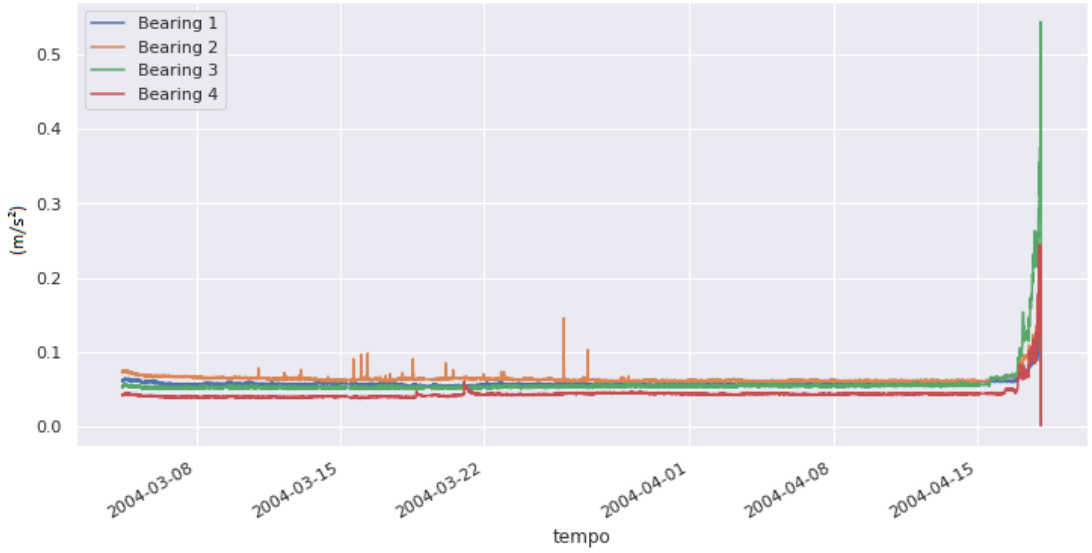


Fonte: Autor.

A segunda forma para obtenção de dados foi a utilização de um *dataset* já estabelecido, disponível em (J. LEE H. QIU; SERVICES, 2007). São utilizados os dados referentes ao terceiro ensaio executado pelos autores. Uma visão do comportamento das amostras pode ser vista em Figura 31. O eixo y representa os valores de aceleração coletados pelos

acelerômetros.

Figura 31 – Visão das amostras do *dataset* utilizado.



Fonte: Autor.

É possível notar que o comportamento dos rolamentos apresenta um comportamento padrão até determinado momento. Na descrição fornecida do *dataset*, a falha total do sistema acontece no último dia de amostragem, no rolamento 3. Entretanto, o padrão visto até o dia 2004-04-15 tem alterações, apresentando um comportamento anômalo de 3 a 4 dias antes da falha acontecer de fato. Para avaliar esse comportamento, as amostras de treinamento do modelo são as coletadas antes do dia 2004-04-11 e, consequentemente, as demais são as de teste. Esse procedimento é necessário, visto que os dados não possuem rótulo, logo a melhor forma de se trabalhar com eles, é definir um intervalo onde o funcionamento do sistema é considerado normal.

Independentemente dos conjuntos de dados utilizados, todos passam por padronização. Uma técnica bastante utilizada é transformar a escala das diferentes variáveis do sistema para uma faixa comum de valores. Assim, para esse trabalho optou-se por utilizar um pré-processamento *StandardScaler*. Esse método tem o intuito de tentar padronizar os dados, removendo a média e escalonando-os para gerar uma dispersão (desvio padrão) unitária. Assim a centralização e o dimensionamento acontecem independentemente em cada tipo de dado do *dataset*, computando as estatísticas relevantes nas amostras no conjunto de treinamento (VALADÃO *et al.*, 2021).

Com as etapas de coleta dos dados e pré-processamento realizadas, a próxima de-

finição está no desenvolvimento do modelo. É necessário estabelecer os atributos do *autoencoder* que serão adotados para a construção do modelo de ML base que será posteriormente otimizado. As camadas da RNA levam em conta a topologia de redes do tipo *autoencoder*, onde uma camada oculta possuirá um afunilamento no número de nós. A quantidade de nós utilizada nesse trabalho estão descritas nas Tabelas 3 e 4.

Tabela 3 – Número de nós por camada para os estudos de caso 1 e 2.

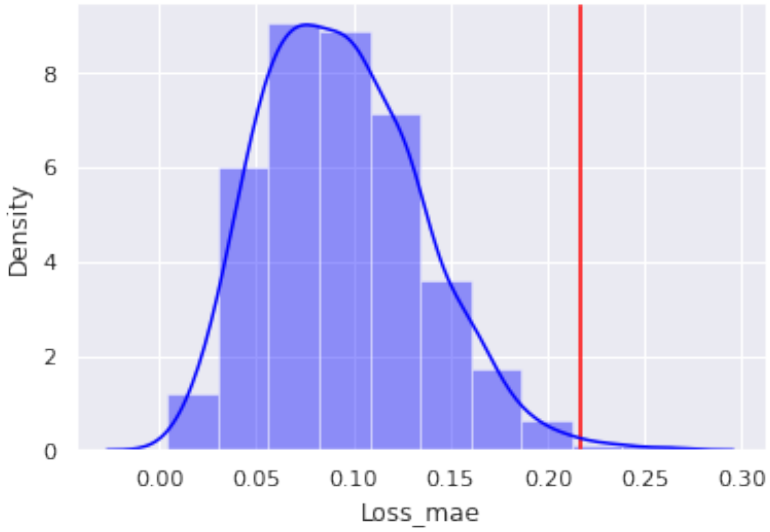
Entrada	Camada 1	Camada 2	Camada 3	Camada 4	Camada 5	Saída
$n\_features\_values$	30	10	2	10	30	$n\_features\_values'$

Note que a Tabela 3 tem como variável o número de entradas, visto que são utilizados cinco conjuntos de dados com quantidades distintas de *inputs*.

Todo o treinamento da rede neural utiliza o *Mean Squared Error* (MSE) como método de avaliação. Para a otimização da rede neural, é utilizado o algoritmo de *Adaptive Moment Estimation*, conhecido como o otimizador $Adam$. Todas as funções de ativação são do tipo *Rectified Linear Unit* (ReLU), e o número de épocas de treinamento foi estipulada em 100, com lotes de 10 amostras. Foi adotado um critério de parada antecipada de treinamento, visando evitar o *overfitting* dos dados de treinamento, definindo em 5 repetições. Após isso, por se tratar de uma rede *autoencoder*, é necessário avaliar a capacidade que a rede adquiriu de reconstruir os dados de entrada. São aplicados então, os mesmos valores de entrada utilizadas no treinamento e se calcula o *Mean Absolute Error* (MAE).

A Figura 32 apresenta a distribuição dos valores de MAE do *dataset* de treinamento com 5 entradas e *offset* = 10. Além da distribuição, já está marcado o valor que será considerado como *threshold* para as amostras de teste. O valor foi gerado conforme a equação 9. Esse é um valor importante, pois ele será o *threshold* adotado para avaliar os dados de teste. Caso a amostra tenho um MAE superior ao valor calculado, será considerada uma amostra com anomalia, caso contrário, será considerada normal.

Figura 32 – Distribuição do MAE da reconstrução dos dados de entrada e a posição do *threshold* destacado em vermelho.



Fonte: Autor.

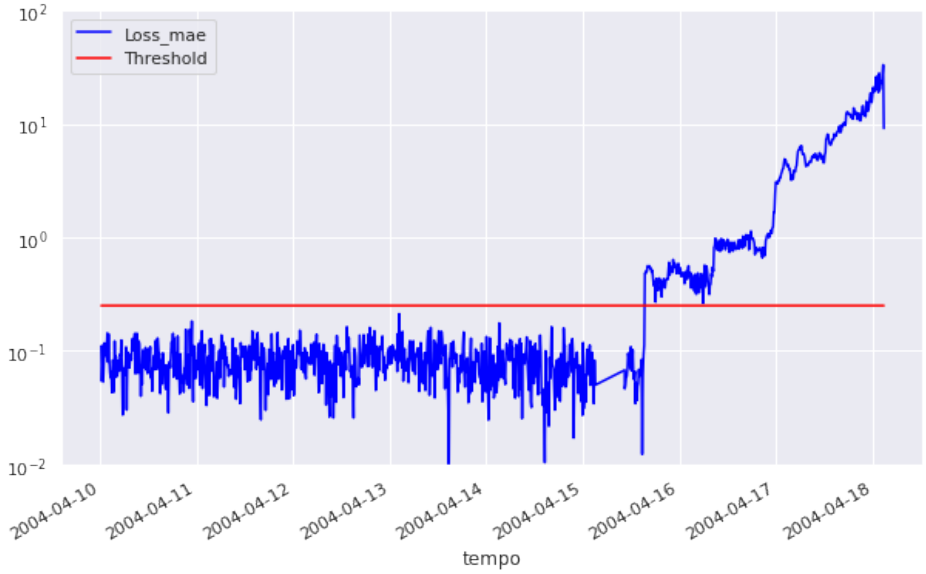
Para o *dataset* da NASA utilizado são realizados os mesmos passos, porém com a rede neural definida na Tabela 4. Como já mencionado, esse conjunto de dados não possui rótulo entre amostras normais e anômalas. Por esse motivo, aplica-se ao modelo treinado nos dados de teste. Na Figura 33 é apresentado o MAE absoluto de cada amostra. É possível perceber que há uma grande diferença nos dados a partir da metade do dia 2004-04-15. Como a falha acontece apenas no último dia amostrado, 2004-04-18, serão adotadas como comportamento normal 768 amostras e anormais 357 amostras. Esse procedimento é necessário para verificar se os modelos otimizados terão o mesmo resultado do modelo não otimizado.

Tabela 4 – Número de nós por camada utilizada no modelo do estudo de caso 3.

Entrada	Camada 1	Camada 2	Camada 3	Camada 4	Saída
4	30	2	30	30	4

Até aqui, todo o desenvolvimento tratou do modelo *TensorFlow* não otimizado, desenvolvido para ser executado em computador. Com esse modelo base criado, são aplicadas as técnicas de otimização, com o auxilio dos recursos disponíveis no *TensorFlow Model Optimization Toolkit*. Para a quantização pós-treino, é preciso calibrar ou estimar o in-

Figura 33 – MAE dos dados de teste.



Fonte: Autor.

tervalo de todos os nós de ponto flutuante no modelo. Ao contrário de nós constantes, como pesos e vieses, tensores variáveis, como entrada de modelo, ativações (saídas de camadas intermediárias) a saída do modelo, não podem ser calibrados a menos que sejam executados alguns ciclos de inferência.

Como resultado, o conversor requer um conjunto de dados representativo para calibrá-los. Esse conjunto de dados pode ser um pequeno subconjunto de cerca de 50-500 amostras dos dados de treinamento ou validação (ABADI *et al.*, 2015). Diferentemente dos demais métodos de otimização, não há necessidade de treinar o modelo novamente. Nas técnicas QAT, *pruning* e agrupamento de pesos, que necessitam de novo treinamento, são adotados os mesmos critérios do modelo base (avaliação de desempenho do treinamento são mantidos inalterados). Na poda de pesos, é adotada uma esparsidade inicial de 0,1 e 0,8 de final. No agrupamento de pesos, são agrupados com K-grupos, onde $K = 4$. O agrupamento é feito pelo método do centroide (API, 2022).

Feita a criação de todos os modelos otimizados, pode-se então converter esses arquivos para modelos TensorFlow Lite Micro, que possibilitam a portabilidade para o sistema embarcado utilizado neste trabalho. É utilizado o comando xxd para realizar a conversão dos modelos, que cria um *dump* hexadecimal de um determinado arquivo ou entrada padrão (WEIGERT, 1997). Um *header* é então gerado, que será anexado à aplicação que posteriormente será gravada no microcontrolador.

```
xxd -i {MODEL_TENSORFLOW} > {MODEL_TFLITE_MICRO}
```

Outra etapa da implementação, é a elaboração do *firmware* para o dispositivo de borda. Como são avaliados dois modos de operação, é preparado na mesma solução de software, um mecanismo de seleção do modo a ser utilizado, o de inferência em *cloud* ou na borda, como pode ser observado na Figura 34. No *firmware* também é armazenado o *dataset* a ser utilizado. Para isso, é criado um *header* com um vetor bi-dimensional MxN, onde M é o número de sinais de entrada que o conjunto de dados dispõe e N a quantidade de amostras que serão avaliadas.

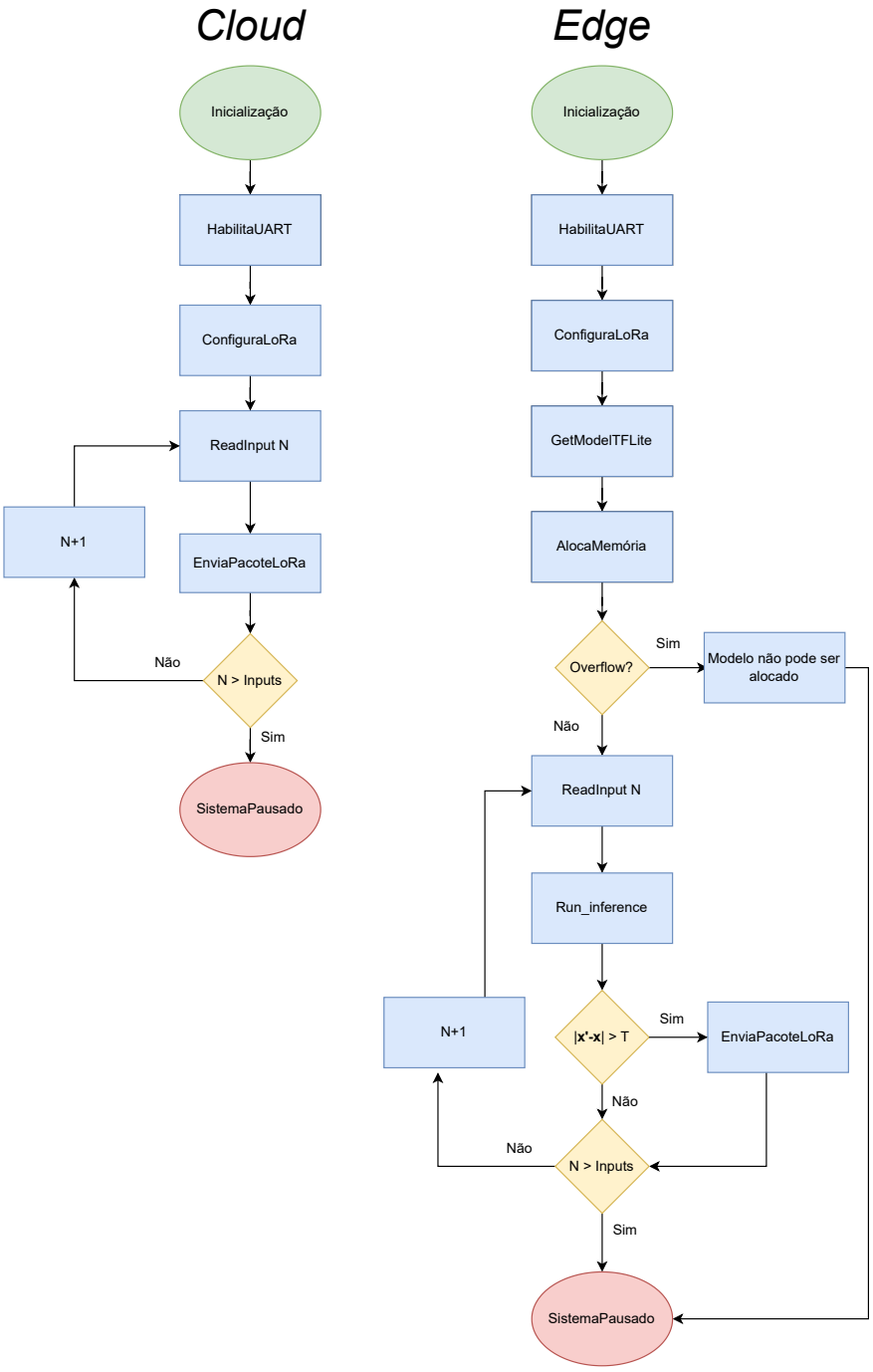
No fluxograma *cloud* está descrito o processo do modo de envio total dos dados. Toda a leitura dos dados salvos no vetor bi-dimensional, são enviados para o módulo LoRa via SPI. O sistema é pausado quando não há mais amostras para serem lidas e enviadas.

Algumas etapas são incluídas para fluxograma *edge*, onde a inferência é realizada localmente. É necessário, na inicialização, alocar espaço na memória SRAM do microcontrolador para o modelo. Caso o vetor exceda os limites de memória, a execução é pausada. Caso haja sucesso na inicialização, é iniciado o mesmo laço de repetição de A, porém, os blocos que realizam a inferência e a condição do resultado encontrado são acrescentados. Caso a diferença absoluta entre o resultado **x'** da reconstrução do sinal de entrada **x** seja maior que o *threshold* definido, é enviado para o coletor qual das entradas possui a maior diferença na reconstrução.

Para coleta dos tempos de execução do microcontrolador é utilizado o *Real Time Clock* (RTC) do RP2040. Ele é programado para realizar contagens de um microssegundo, fornecendo a resolução necessária para a verificação do tempo de cada uma das inferências realizadas na borda.

Como última etapa da implementação, está o desenvolvimento da aplicação do dispositivo coletor e da aplicação que irá exibir e inferir os dados apurados. O *firmware* utilizado no coletor é integralmente o utilizado por (VALADÃO *et al.*, 2021). Já a aplicação que faz a leitura dos dados recebidos pelo coletor utilizado no trabalho foi adaptada de (VALADÃO *et al.*, 2021). Ela é executada em computador e recebe todos os dados do dispositivo coletor via USB-Serial. Toda aplicação está escrita em $python$. Foi acrescentada a essa rotina de coleta de dados, uma chamada para realizar a inferência dos dados recebidos. Toda vez que um conjunto de dados chega pela serial, é feito o *parse* para remover as informações do cabeçalho e utilizar o *payload* que contém as entradas para o

Figura 34 – Fluxogramas do software embarcado para os dois modos de operação.

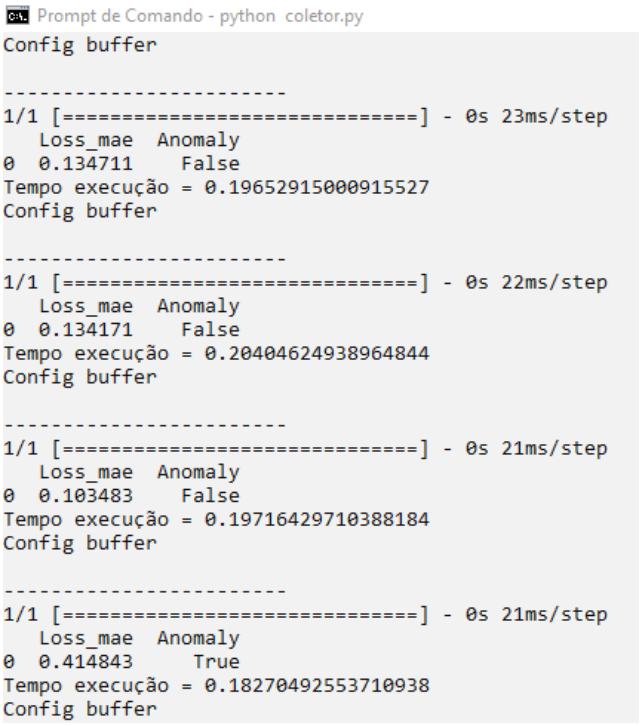


Fonte: Autor.

autoencoder. Como a execução é realizada em um computador, o modelo não passa por otimizações e está em formato $HDF5$. O arquivo $HDF5$ significa *Hierarchical Data Format 5*. É um arquivo útil para armazenar uma grande quantidade de dados e como o nome sugere, ele armazena dados em uma estrutura hierárquica dentro de um único arquivo. Para a avaliação de alguns tempos de execução no modo *cloud*, é utilizada a

biblioteca $time$. Os momentos de coleta de tempo são: recebimento de pacote via USB-Serial, *parser* dos dados e inferência, como pode ser observado em um exemplo da Figura 35. Todos os valores recebidos, os tempos e o MAE de cada inferência são armazenados em um arquivo CSV para posterior conferência e gerar as visualizações.

Figura 35 – Visão dos logs da execução da aplicação que recebe os dados via USB-SERIAL, realiza o *parse* e a inferência.



Fonte: Autor.

6 ESTUDOS DE CASO

Os três estudos de caso são apresentados para a avaliação completa das técnicas de otimização em modelos de ML. Os estudos têm como objetivo a validação do sistema proposto, além de obter resultados que auxiliem na tomada de decisão em futuras aplicações dentro da IoT. Para a avaliação dos consumos de energia nos diferentes modos de operação, foi estabelecido um *setup* descrito em 4.3.

Para melhor visualização, atribuiu-se a seguinte legenda para cada técnica de otimização dos modelos de *TinyML*:

- A1: Não otimizado;
- A2: Quantização pós-treino;
- A3: Treinamento consciente de quantização;
- A4: *Pruning* (poda de pesos);
- A5: Agrupamento de pesos.

6.1 Estudo de Caso 1 - Tempo de Inferência e Tamanhos de Compressão

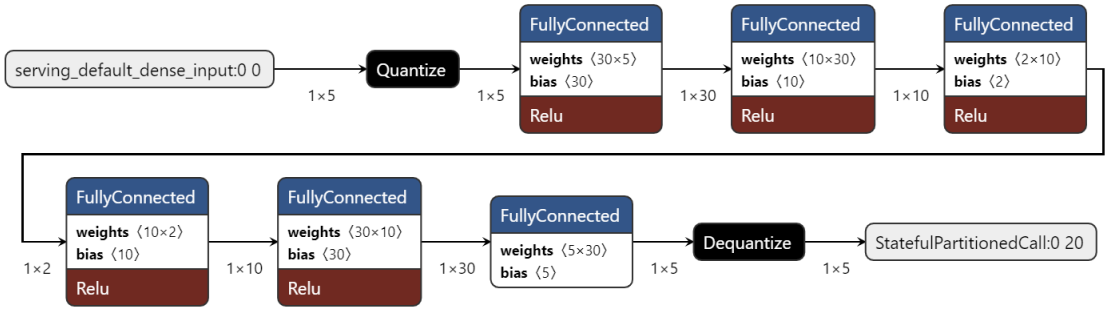
Nesta seção é apresentado um estudo de caso onde são gerados 5 diferentes *datasets*, variando a quantidade caracterísiticas dos sinais de entrada: 5, 10, 15, 20 e 25. Os conjuntos de dados são gerados com $offset$ = 10 – descrito na seção 4.4. A Tabela 5 apresenta os valores em *bytes* do espaço necessário para cada modelo *TinyML*. Esses são os valores do arquivo gerado na conversão do modelo *TensorFlow* para *TensorFlow Lite*.

Tabela 5 – Valor em bytes dos tamanhos do modelos de *TinyML* otimizados pelas técnicas avaliadas.

Entradas	A1	A2	A3	A4	A5
5	7024	4648	5632	6808	6764
10	8288	5008	6064	8084	8036
15	9552	5368	6392	9340	9292
20	10768	5688	6712	10560	10512
25	12000	6008	7040	11780	11732

As técnicas A2 e A3 apresentam as maiores reduções em relação ao modelo A1. No caso de A2 a redução de tamanho é esperada, pois está se convertendo pesos da rede neural que são variáveis do tipo *float*, que ocupam 4 bytes em memória, por pesos do tipo *int8* (que ocupam apenas 1 byte). Neste caso, o fato de não se ter uma redução ainda maior está no formato do arquivo que é gerado, que necessita armazenar alguns parâmetros da quantização, como as valores da equação (1). Já para A3, são anexados a esse arquivo, além dos pesos, dois blocos que fazem a quantização dos valores de entrada e a desquantização na saída, como mostra a Figura 36.

Figura 36 – Estrutura do modelo otimizado pela técnica A3 (QAT).



Fonte: Autor.

Os modelos A4 e A5 também tem redução em relação a A1, porém em menor proporção. Esse fato se deve a forma que as técnicas realizam a otimização. No caso de A4, a poda de pesos não é capaz de reduzir o tamanho do modelo em termos de espaço ocupado. A poda encontra os pesos menos significativos dos modelos (aqueles que estão próximos de zero) e os força a zero. Ao salvar um modelo, os pesos ocupam o mesmo espaço, independentemente de seus valores serem zero ou não.

Para evidenciar as constatações acerca de cada técnica, é elucidativo olhar para os pesos de uma camada da rede neural. A Tabala 6 mostra os pesos de uma das camadas ocultas do *autoencoder*, usando a parte de *encode*, das camadas com 10 nós para a camada com 2 nós. Em cada uma das colunas é apresentado o valor do peso para a respectiva técnica.

Tabela 6 – Pesos para cada técnica de otimização, olhando para uma das camadas da RNA.

	A1	A2	A3	A4	A5
	-0,290755	-54	-54	0,000000	0,021452
	0,314913	59	57	0,491734	0,512354
	-0,323560	-60	-61	0,000000	0,021452
	-0,224437	-42	-34	0,000000	0,021452
Camada 10x2	0,052022	10	10	0,000000	0,021452
Conexão 1	0,226106	42	40	0,000000	0,021452
	0,357375	66	67	0,000000	0,021452
	-0,050444	-9	-9	0,000000	0,021452
	0,271387	50	49	0,000000	0,021452
	0,186117	35	34	0,000000	0,021452
	0,044677	8	8	0,000000	0,021452
	-0,076111	-14	-14	0,000000	0,021452
	0,461422	86	86	0,000000	0,021452
	0,270206	50	47	0,000000	0,021452
Camada 10x2	0,657090	122	123	0,657090	0,800867
Conexão 2	0,627599	117	115	0,838144	0,800867
	-0,160195	-30	-30	0,000000	0,021452
	0,306654	57	57	0,000000	0,021452
	0,683059	127	127	0,702237	0,512354
	-0,360523	-67	-67	0,000000	0,021452

Como mencionado, A2 e A3 possuem valores muito similares. Em A4, nota-se que muitos dos valores estão zerados, algo esperado na técnica de poda de pesos. Por fim, em A5 é possível verificar os K valores de centroides definidos. É possível aplicar algoritmos

de compactação, como Gzip (LINUXISE, 2019), que são eficientes em dados que contêm valores semelhantes, casos de A4 e A5.

A compactação de um modelo é útil para reduzir o tamanho de um aplicativo móvel que contém o modelo ou reduzir a banda ao enviá-lo pela rede. Neste caso, o modelo pode ser descomprimido (LOUP GAILLY; FOUNDATION, 2022) antes de ser executado. Na tabela 7 estão os valores dos mesmos modelos anteriores, porém compactados pela ferramenta Gzip. Olhando por essa perspectiva, as técnicas que resultam em pesos com valores que possam se repetir, tem ampla vantagem quando se considera o tamanho do vetor final.

Tabela 7 – Valor em bytes dos tamanhos dos modelos de ML otimizados pelas técnicas avaliadas e posteriormente compactadas pela ferramenta *gzip*.

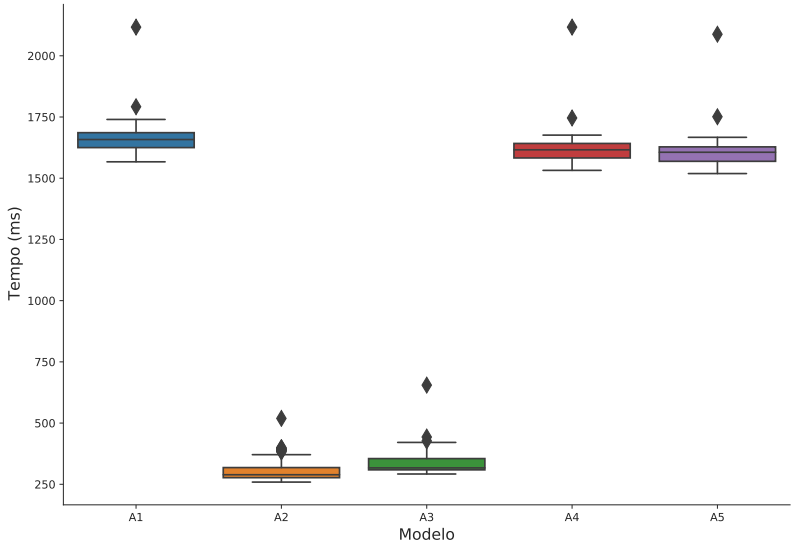
Entradas	A1	A2	A3	A4	A5
5	5070	2575	2804	2710	1973
10	6193	2909	3135	3102	2103
15	7343	3217	3459	3458	2188
20	8463	3509	3768	3817	2273
25	9598	3854	4082	4176	2414

Esse tipo de compactação pode ser extremamente útil para aplicações venham a atualizar seus modelos *Over-The-Air* (OTA). Em SUDHARSAN *et al.* (2022) os autores aplicaram a transferência do modelo de ML via HTTPS para um hardware que realiza detecção de anomalia na borda. Entretanto, eles não utilizaram nenhuma técnica de otimização nos seus modelos.

Outro aspecto avaliado é o tempo de execução por inferência realizada. A seguir, as figuras 37, 38, 39, 40 e 41 apresentam os diagramas de caixa para cada conjunto de dados avaliados (5, 10, 15, 20, 25 características de entrada). O mesmo padrão é percebido, onde as técnicas que utilizam quantização têm o tempo de inferência muito menor, visto que não utilizam ponto flutuante em seus cálculos. Os resultados obtidos foram gerados após a execução da aplicação em uma Raspberry Pi Pico.

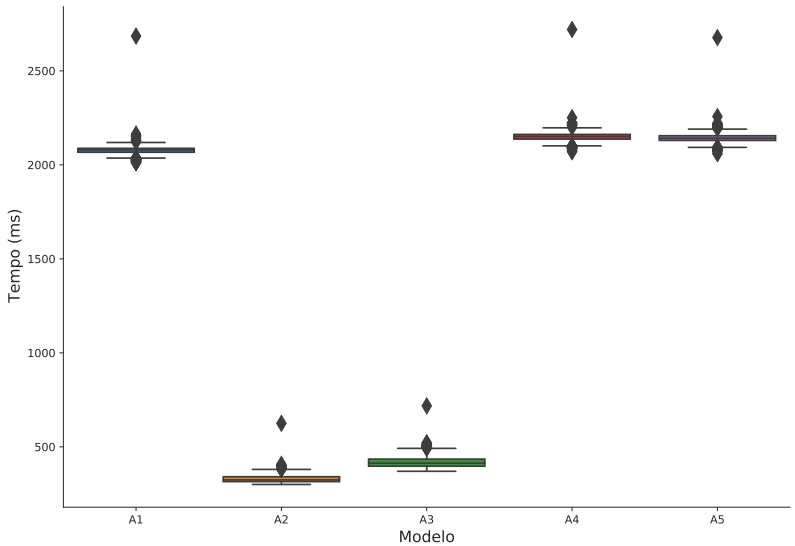
Em todas as figuras acima, a variação de tempo dentro da técnica para cada conjunto de dados se mantém estável, tendo distribuição bastante homogênea. Destaca-se que os pontos fora dos intervalos padrões, são causados pela primeira inferência realizada em

Figura 37 – Diagrama de caixa dos tempos coletados em milissegundos para os 5 modelos avaliados considerando 5 características de entrada.



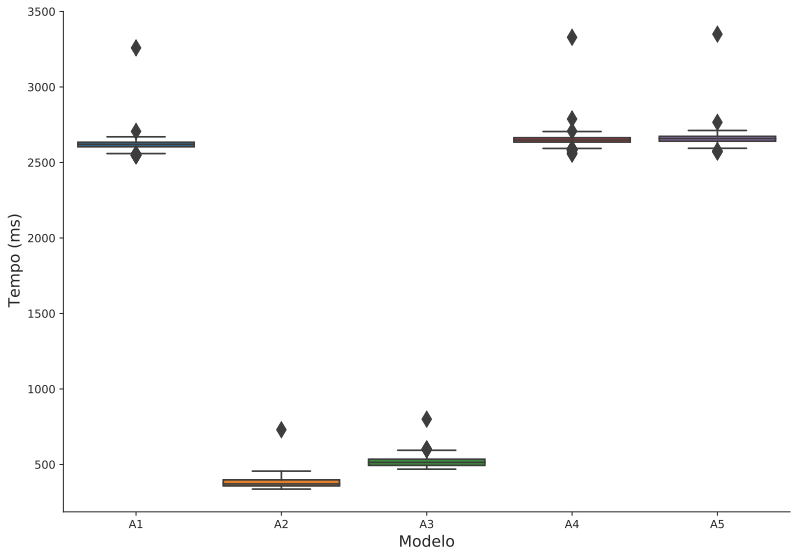
Fonte: Autor.

Figura 38 – Diagrama de caixa dos tempos coletados em milissegundos para os 5 modelos avaliados considerando 10 sinais de entrada.



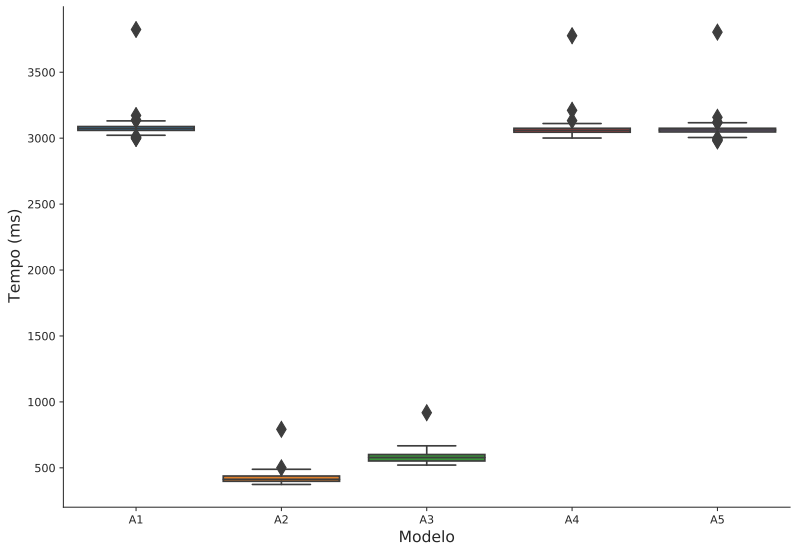
Fonte: Autor.

Figura 39 – Diagrama de caixa dos tempos coletados em milissegundos para os 5 modelos avaliados considerando 15 sinais de entrada.



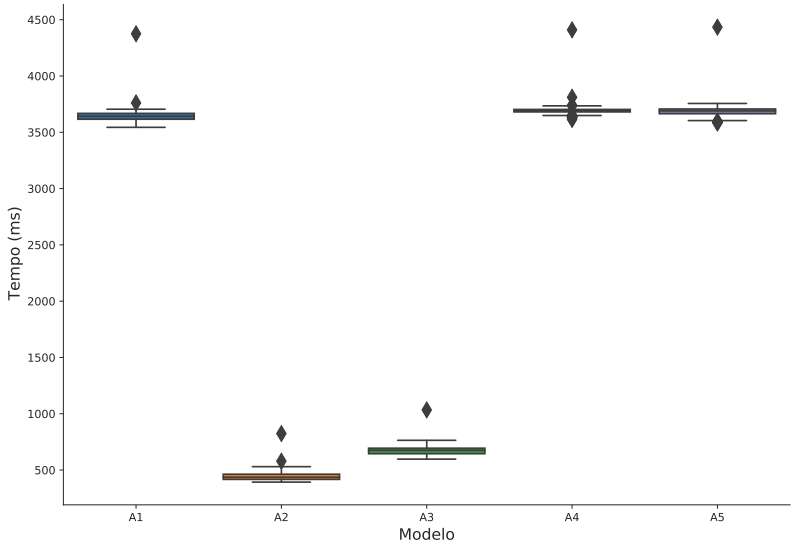
Fonte: Autor.

Figura 40 – Diagrama de caixa dos tempos coletados em milissegundos para os 5 modelos avaliados considerando 20 sinais de entrada.



Fonte: Autor.

Figura 41 – Diagrama de caixa dos tempos coletados em milissegundos para os 5 modelos avaliados considerando 25 sinais de entrada.



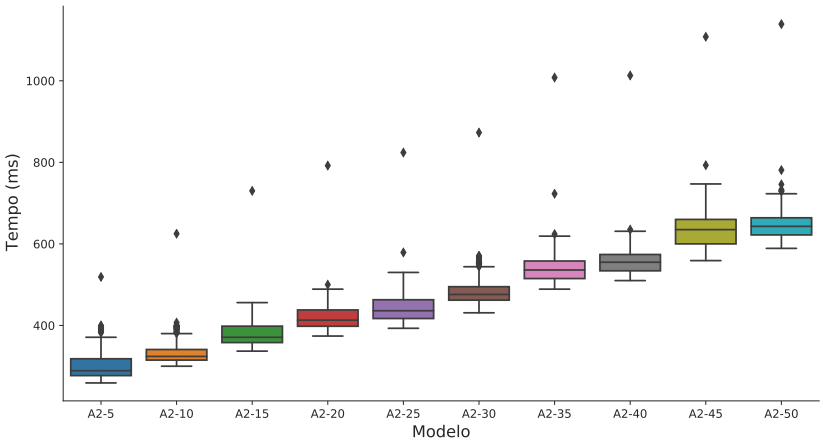
Fonte: Autor.

cada *dataset*, onde os tensores ainda não alocaram o seu espaço na SRAM do microcontrolador – esse processo é diferente da alocação de espaço para o modelo, descrito na Figura 34.

Como no trabalho de BANBURY *et al.* (2020) foi verificado que a latência crescia linearmente com o aumento do número de operações, decidiu-se verificar se esse fato se aplica para as técnicas de otimização. Na Figura 42 foram gerados mais alguns conjuntos de dados para verificar a existência dessa constatação.

É possível perceber um comportamento linear a medida que mais operações são incluídas na rede. Como o crescimento do número de entradas tem um passo de 5, o número de operações é acrescido de mais 5 nós de saída, pois a topologia adotada é a do *autoencoder*.

Figura 42 – Diagrama de caixa dos tempos coletados em milissegundos para a técnica A2 em diferentes quantidades de entradas dos conjuntos de dados.



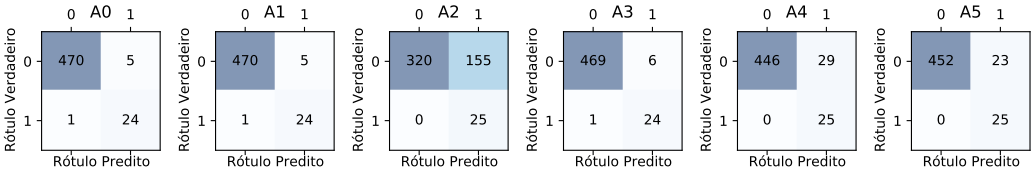
Fonte: Autor.

6.2 Estudo de Caso 2 - Desempenho

Este estudo de caso apresenta uma análise da desempenho e qualidade nos modelos otimizados pelas técnicas utilizadas. No estudo, são gerados 5 diferentes *datasets*, variando a quantidade de sinais de entrada: 5, 10, 15, 20 e 25, e os conjuntos de dados são gerados com *offset* = 5. Como está se mantendo a mesma estrutura da rede, os valores de tamanhos permanecem os mesmos.

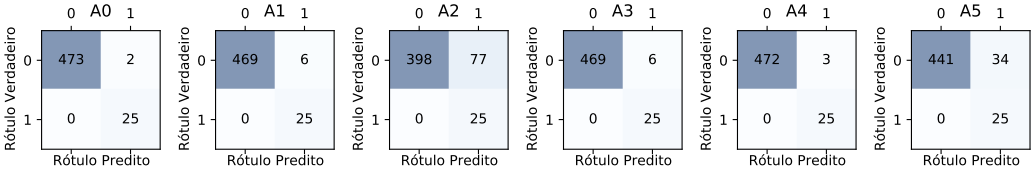
A diferença para o estudo anterior está na redução da distribuição dos valores dos conjuntos de dados gerados (*offset* de 10 para 5). Essa diferença irá deixar a distribuição dos dados sintéticos gerados mais ajustada, variando em um intervalo muito menor. São acrescentados aos resultados das técnicas de otimização, os resultados obtidos quando a inferência é realizada em um modelo de ML não-otimizado, executado em PC, legendados como A0. As figuras 43, 44, 45, 46 e 47 apresentam as matrizes confusão para cada conjunto de dados avaliado. Para as amostras consideradas normais, foi atribuído o rótulo 0 e, consequentemente, as amostras anômalas possuem o rótulo 1.

Figura 43 – Matriz Confusão para 5 sinais de entrada.



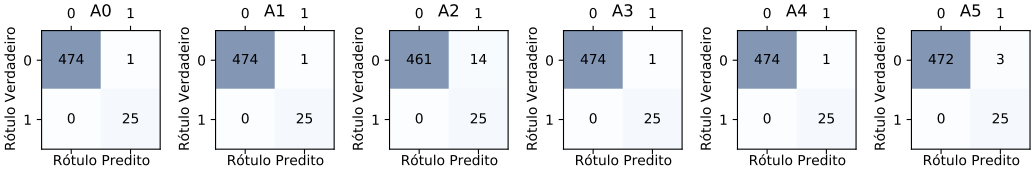
Fonte: Autor.

Figura 44 – Matriz Confusão para 10 sinais de entrada.



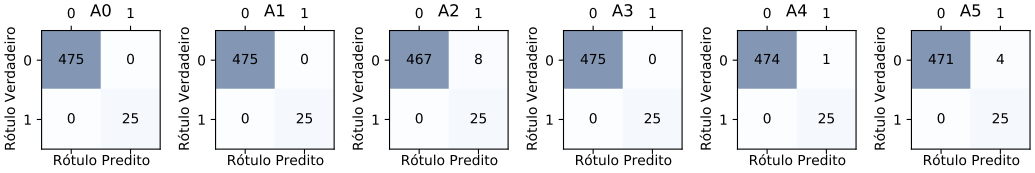
Fonte: Autor.

Figura 45 – Matriz Confusão para 15 sinais de entrada.



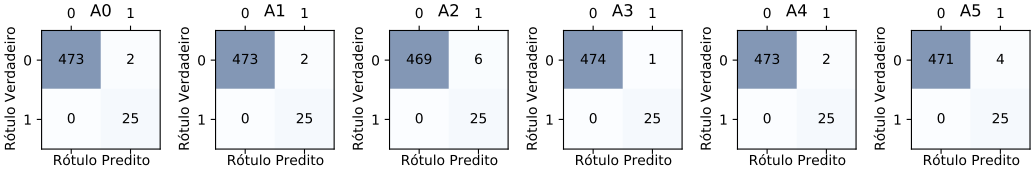
Fonte: Autor.

Figura 46 – Matriz Confusão para 20 sinais de entrada.



Fonte: Autor.

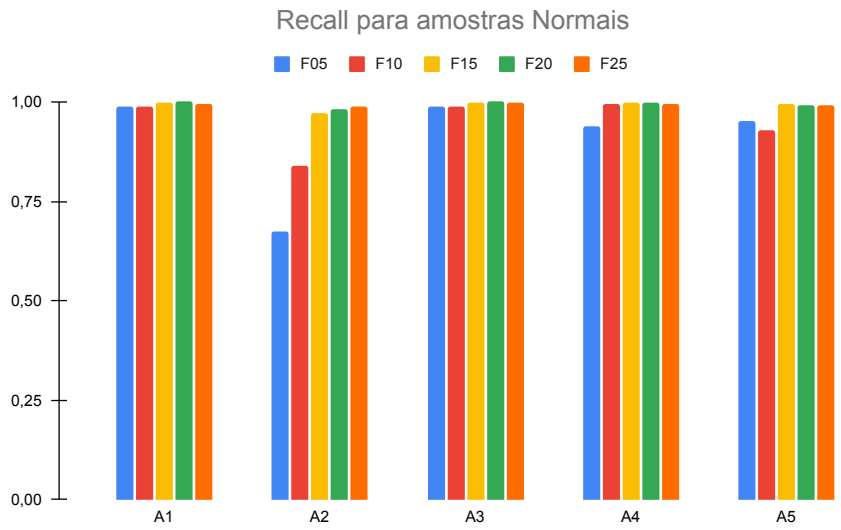
Figura 47 – Matriz Confusão para 25 sinais de entrada.



Fonte: Autor.

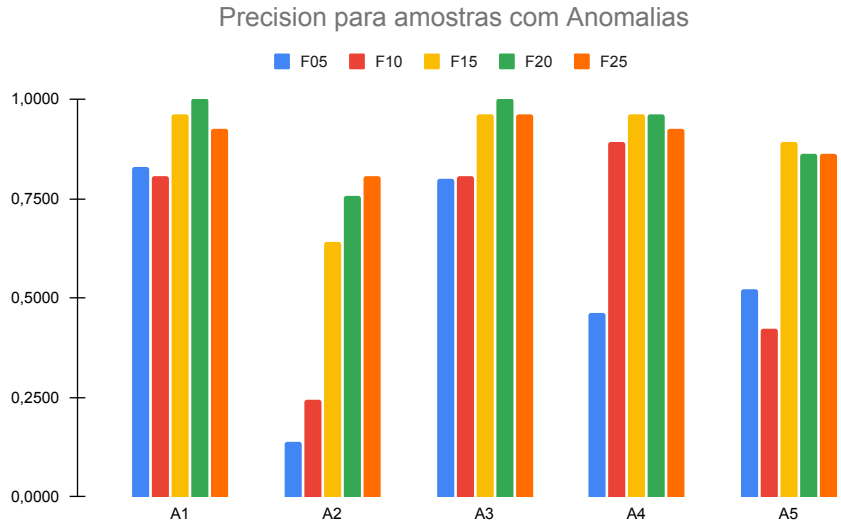
Utilizando os valores obtidos, é possível calcular as métricas de desempenho (*Precison*, *Recall* e $F1_{Score}$). As figuras 48 e 49 apresentam graficamente a evolução do número de entradas para cada uma das técnicas, olham para algumas métricas que mais se destacaram para cada uma das classes de amostras.

Figura 48 – *Recall* para amostras normais.



Fonte: Autor.

Figura 49 – *Precision* para amostras com anomalias.



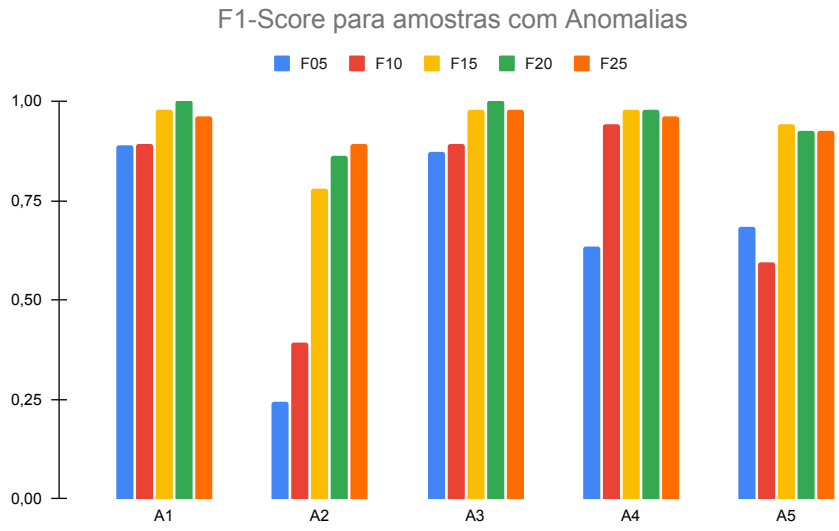
Fonte: Autor.

Nota-se que há um bom desempenho do *autoencoder* otimizados para definir as amostras normais, exceto na técnica de otimização pós-treinamento com 5 e 10 entradas. Ao verificar o desempenho para a classe de amostras com anomalias, é possível perceber que a medida que mais sinais de entradas são utilizados, os modelos de ML como um todo tem uma melhora significativa.

Avaliando as técnicas de forma individual, destaca-se a técnica A3, de QAT, uma vez que, para todos os *datasets* avaliados, é possível notar que seu desempenho é igual ou superior a técnica A1 e ao modelo executado em um PC. Novamente, os piores resultados estão na técnica A2. Ao explorar em detalhes as reconstruções feitas por ela, foi possível verificar que cerca 35% os FP para o *dataset* com 5 entradas ficaram muito próximos ao *threshold*. Essa técnica apresenta o maior *trade-off*, algo esperado, visto que os pesos são simplesmente convertidos de variáveis com 32 bits para 8 bits, perdendo resolução.

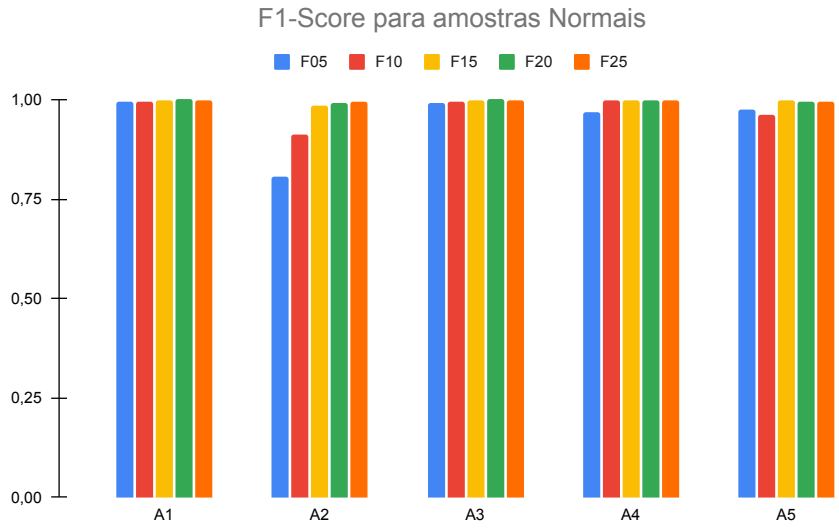
Para observar o desempenho geral das redes *autoencoder* otimizadas, são plotados os valores de $F1_{Score}$ nas figuras 50 e 51.

Figura 50 – $F1_{Score}$ para amostras com anomalias.



Fonte: Autor.

Figura 51 – $F1_{Score}$ para amostras com normais.



Fonte: Autor.

A Tabela 8 apresenta o compilado de todos esses valores para cada uma das execuções que foram realizadas no hardware adotado, trazendo todos os valores calculados para as métricas de desempenho.

Tabela 8 – *Precision, Recall* e *F1-Score* para cada *dataset* e técnica avaliados.

Features - Técnica	Classe	*Precision*	*Recall*	$F1_{Score}$	*Support*	*Accuracy*
F5 - A1	Normal	0,998	0,989	0,994	475	0,988
	Anomalia	0,828	0,960	0,889	25	
F5 - A2	Normal	1,000	0,674	0,805	475	0,690
	Anomalia	0,139	1,000	0,244	25	
F5 - A3	Normal	0,998	0,987	0,993	475	0,986
	Anomalia	0,800	0,960	0,873	25	
F5 - A4	Normal	1,000	0,939	0,969	475	0,942
	Anomalia	0,463	1,000	0,633	25	
F5 - A5	Normal	1,000	0,952	0,975	475	0,954
	Anomalia	0,521	1,000	0,685	25	
F10 - A1	Normal	1,000	0,987	0,994	475	0,988
	Anomalia	0,806	1,000	0,893	25	
F10 - A2	Normal	1,000	0,838	0,912	475	0,846
	Anomalia	0,245	1,000	0,394	25	
F10 - A3	Normal	1,000	0,987	0,994	475	0,988
	Anomalia	0,806	1,000	0,893	25	
F10 - A4	Normal	1,000	0,994	0,997	475	0,994
	Anomalia	0,893	1,000	0,943	25	
F10 - A5	Normal	1,000	0,928	0,963	475	0,932
	Anomalia	0,424	1,000	0,595	25	
F15 - A1	Normal	1,000	0,998	0,999	475	0,998
	Anomalia	0,962	1,000	0,980	25	
F15 - A2	Normal	1,000	0,971	0,985	475	0,972
	Anomalia	0,641	1,000	0,781	25	
F15 - A3	Normal	1,000	0,998	0,999	475	0,998
	Anomalia	0,962	1,000	0,980	25	
F15 - A4	Normal	1,000	0,998	0,999	475	0,998
	Anomalia	0,962	1,000	0,980	25	
F15 - A5	Normal	1,000	0,994	0,997	475	0,994
	Anomalia	0,893	1,000	0,943	25	
F20 - A1	Normal	1,000	1,000	1,000	475	1,000
	Anomalia	1,000	1,000	1,000	25	
F20 - A2	Normal	1,000	0,983	0,992	475	0,984
	Anomalia	0,758	1,000	0,862	25	
F20 - A3	Normal	1,000	1,000	1,000	475	1,000
	Anomalia	1,000	1,000	1,000	25	
F20 - A4	Normal	1,000	0,998	0,999	475	0,998
	Anomalia	0,962	1,000	0,980	25	
F20 - A5	Normal	1,000	0,992	0,996	475	0,992
	Anomalia	0,862	1,000	0,926	25	
F25 - A1	Normal	1,000	0,996	0,998	475	0,996
	Anomalia	0,926	1,000	0,962	25	
F25 - A2	Normal	1,000	0,987	0,994	475	0,988
	Anomalia	0,806	1,000	0,893	25	
F25 - A3	Normal	1,000	0,998	0,999	475	0,998
	Anomalia	0,962	1,000	0,980	25	
F25 - A4	Normal	1,000	0,996	0,998	475	0,996
	Anomalia	0,926	1,000	0,962	25	
F25 - A5	Normal	1,000	0,992	0,996	475	0,992
	Anomalia	0,862	1,000	0,926	25	

6.3 Estudo de Caso 3 - Avaliação do Sistema

Os estudos de caso apresentados anteriormente nas seções 6.1 e 6.2 trouxeram uma avaliação das técnicas para conjuntos de dados gerados por meio de uma ferramenta. Foram observadas a capacidade de redução do tamanho e tempo de inferência de cada modelo, bem como o desempenho de cada técnica. Neste estudo de caso, é utilizado um *dataset* de uma aplicação real (J. LEE H. QIU; SERVICES, 2007). Os resultados são apresentados com duas visões distintas. Na subseção 6.3.1, são apresentadas as comparações entre as técnicas de otimização, já a Subseção 6.3.2 apresenta as métricas de desempenho da arquitetura proposta.

6.3.1 Modelos Otimizados

A Tabela 9 apresenta os resultados de redução encontrados para cada uma das técnicas de otimização avaliadas.

Tabela 9 – Tamanho final de cada técnica de otimização.

Técnicas de Otimização	Tamanho Modelo CC (Bytes)	Tamanho Compactado Gzip (Bytes)
A1	7984	6214
A2	4560	2713
A3	5520	2842
A4	7894	3054
A5	7760	1978

A coluna Tamanho Modelo CC apresenta o tamanho final do modelo (vetor) gerado pela ferramenta do TensorFlow Lite. O modelo A2 é notoriamente o menor entre todos, visto que a quantização pós-treino é a mais simplificada entre todas as técnicas - convertendo variáveis do modelo de 32 bits para inteiros de 8 bits. Em A3 também há significativa redução no vetor. Em A4 e A5 há pouca redução em relação a A1. Para o caso da poda de pesos, isso é facilmente explicado pelo fato de que, nessa técnica os pesos menos significativos venham a convergir para zero, ainda haverá a ocupação do byte no vetor. Há semelhança para o agrupamento de pesos (A5), já que o vetor irá conter agora os centroides dos *clusters* e irá atribuir um índice para cada peso do vetor de pesos do

modelo.

A grande diferença para esses dois modelos (A4 e A5) está apresentada na coluna Tamanho Modelo Gzip. Ferramentas de compactação (como gzip) aproveitam as redundâncias nos dados para obter uma compactação mais efetiva. Ambos os modelos facilitariam, por exemplo, a transferência de uma atualização de software para o dispositivo de borda, visto que o modelo é bastante reduzido. Quanto mais complexa a rede for, maior será a a percepção deste fato (HAN; MAO; DALLY, 2016).

Os modelos A2 e A3 também têm significativa redução quando compactados, o que é explicado pelo fato da possível repetição de valores inteiros no vetor - acontecimento muito mais provável do que se fossem do tipo **float**.

Na Tabela 10 são mostrados os valores para as métricas de desempenho. Em A1, modelo TensorFlow Lite obteve exatamente o mesmo resultado do modelo TensorFlow desenvolvido e explicado ilustrado na Figura 33. Esse resultado por si só já é inspirador, visto que é possível afirmar que a inferência na borda não impacta nenhuma métrica de eficiência na detecção de casos de anomalia, quando comparado ao modelo baseado em nuvem.

Tabela 10 – Resultado dos modelos otimizados.

Técnica de Otimização	Recall	Precision	$F1_{Score}$	Acuracy
A1	1,0000	1,0000	1,0000	1,0000
A2	1,0000	0,9107	0,9533	0,9689
A3	1,0000	0,9972	0,9986	0,9991
A4	1,0000	0,9972	0,9986	0,9991
A5	0,9832	1,0000	0,9915	0,9947

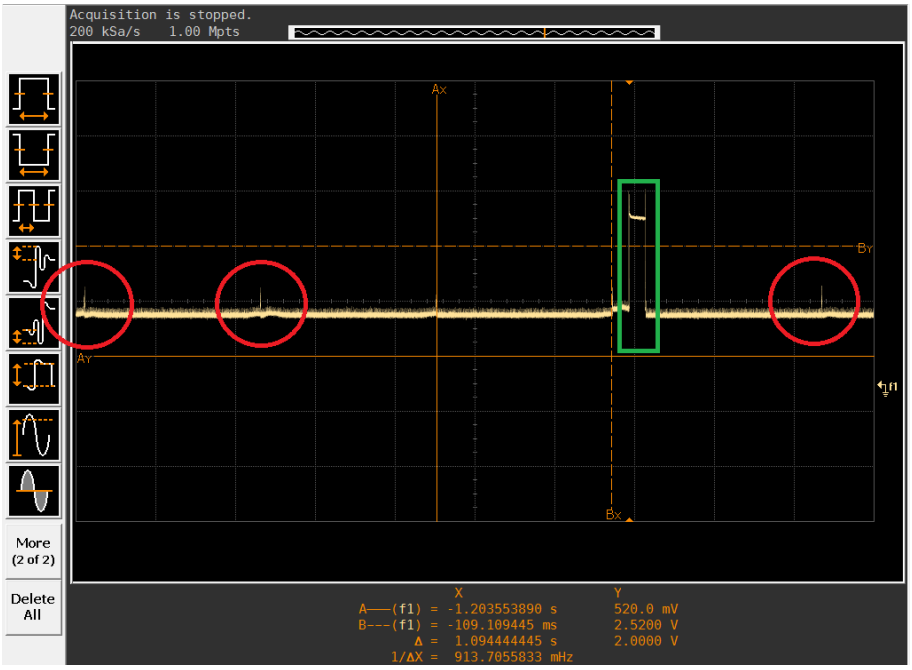
O modelo que teve o pior desempenho, ainda que bastante próximo, foi o modelo A2 – fato observado no estudo de caso 2. Em todos os modelos, houve a mesma detecção do primeiro VP - amostra de número 769.

6.3.2 Arquitetura

Como não foram constatadas diferenças no consumo energético do dispositivo quando estavam sendo executados os demais modelos, será apresentado apenas o resultado para

modelo não otimizado A1. Visto que não estão sendo avaliadas as características do modelo, mas sim a solução de inferência na borda proposta, o único fator que poderia apresentar uma divergência, seria no momento da inferência. Na Figura 52, é possível notar pequenos picos de tensão, destacados em vermelho, cadenciados por 1s, caracterizados pelo processo de leitura de uma amostra e sua inferência no modelo de *TinyML*. O momento da detecção de uma amostra anômala também é visível, visto que após detectar uma amostra com anomalia é feita a transmissão, destacada em verde, informando o rolamento que apresentou a falha.

Figura 52 – Captura do osciloscópio que retrata o processo de inferência no dispositivo de borda.



Fonte: Autor.

Na Tabela 11 estão as informações relevantes para análise desse resultado. A quantidade de envios corresponde, no modo de operação sem inferência na borda, ao número total de amostras disponíveis. No modo com inferência no embarcado da borda, todas as 357 amostras classificadas como anômalos são envidadas ao coletor.

Em ambos os modos, o mesmo cabeçalho - 12 bytes - com informações é anexado ao *payload*. A diferença entre ambos está na carga útil: é necessário enviar 4 valores de 4 bytes cada, pois a inferência é baseada no em sistema *cloud* e os dados são do tipo *float*. Já para a inferência no hardware da borda, destinam-se dois bytes para informar o rolamento com problema. Esses dois fatores combinados, resultaram em uma redução de

Tabela 11 – Comparação entre a operação com e sem inferência na borda, considerando apenas o processo de transmissão.

	Sem inferência	Com inferência
Quantidade de envios	1125	357
Tamanho pacote (bytes)	28	14
Tempo envio (ms)	133	103
Energia por envio (mJ)	5,497	4,257
Energia total (J)	6,183	1,519

quase 4 vezes no consumo, resultado muito simular ao observado em MOALLEMI *et al.* (2022), onde os autores conseguiram reduzir o consumo da aplicação em 5 vezes ao levar a inferência para a borda do sistema. Um possível *trade-off* causado por ter o modelo dentro do sistema embarcado foi descartado após não ser percebido um consumo de energia significativamente diferente quando executando a inferência na borda. Em aplicações que tenham uma frequência maior de envio ou uma quantidade maior de bytes por envio, essa diferença tende a aumentar ainda mais a favor de soluções com inferência na borda. Outra avaliação está no tempo de inferência de ambos os modos de operação. Considerando o diagrama de sequência apresentado na seção 5.1, foi realizado o levantamento do tempo que cada etapa do modo *cloud* levaria para informar o dispositivo de borda da inferência. Os dados desse levantamento estão descritos na Tabela 12.

É possível reduzir o tempo em que a inferência fica disponível para o dispositivo de borda, baseado nessa aplicação, de 423,9 milissegundos para apenas 2,3 milissegundos, redução de mais de 99% – sem considerar questões subjetivas, como segurança da informação. Levando em considerações os resultados de desempenho obtidos por todas as técnicas de inferência na borda, a arquitetura se mostra robusta e com vantagens claras.

Tabela 12 – Tempos por etapas para a realização de uma inferência.

Modo Operação	Tempo TX (ms)	Tempo USB-Serial (ms)	Tempo Inferência (ms)	Tempo RX* (ms)	Total (ms)
Cloud	133	0,9	187	103	423,9
Edge	-	-	2,3	-	2,3

* A implementação de RX não foi executada, porém, utilizou-se o tempo do menor envio utilizado no trabalho como o valor a ser adotado.

7 CONCLUSÕES

As aplicações de IoT abrangem diversos propósitos de utilização, incluindo a detecção de anomalias. Avaliar se um comportamento está dentro de um determinado padrão em um ambiente industrial, por exemplo, pode exigir respostas em tempo real e baixo consumo de energia. Olhando para esse contexto, torna-se necessária a discussão de soluções que envolvam o cenário de tomada de decisão na borda dos sistemas.

O trabalho apresentado foi motivado na necessidade de se avaliar técnicas de otimização para modelos de *TinyML*, a fim de estabelecer alguns cenários de análise de detecção de anomalias em diferentes conjuntos de dados. Uma esteira de desenvolvimento é proposta, onde são descritas as etapas para a construção de uma solução que embarque um modelo de ML em um hardware com baixos recursos computacionais.

Estudos de caso são utilizados para comparar aspectos de compactação e performance dos modelos otimizados. Na avaliação das técnicas de otimização, foi possível perceber a redução direta do tamanho dos vetores utilizados no *firmware* para os métodos que envolvem quantização. Entretanto, quando utilizada uma ferramenta de compactação, as técnicas de *pruning* e agrupamento de pesos se sobressaem, haja vista possibilidade de se agrupar as matrizes esparsas e os valores dos centroides do agrupamento, respectivamente.

Além disso, foram verificados os tempos de execução para 5 conjuntos de dados, variando a quantidade de entradas. As técnicas de quantização tem a menor latência, pois a maioria dos cálculos na rede neural não envolvem ponto flutuante. Quando avaliada a qualidade de cada modelo, é possível perceber que a utilização das técnicas não impacta de forma negativa na detecção

Para complementar a análise, são apresentados também os resultados e métodos para avaliar uma aplicação IoT que utiliza dados de um sistema real do monitoramento de

rolamentos. É estabelecida a utilização de uma LPWAN para comparar dois modos de operação. As vantagens de se realizar a inferência na borda do sistema são comprovadas. Nos testes realizados, o consumo de energia foi reduzido em 75% e o tempo de diagnóstico de cada amostra em mais de 99%.

7.1 Contribuições

O paradigma TinyML ainda está em seu estágio inicial, que requer alinhamentos adequados para se adaptar às estruturas de IoT existentes. O trabalho busca apresentar alternativas para hardwares com baixos recursos de processamento computacional, avaliando aspectos que algumas técnicas de otimização podem propiciar. Além disso, é realizada a criação de uma esteira de desenvolvimento para embarcar os modelos de ML otimizados. Com carência na literatura de estudos que realizasse a comparação entre os métodos de otimização, essa dissertação colabora com resultados consistentes e facilmente replicáveis por toda a comunidade científica, visto que todos os métodos e materiais, bem como a implementação foram criteriosamente descritos.

É importante ressaltar que foram caracterizados vários aspectos, como tamanho final de cada modelo otimizado, tempo para realizar inferência e aspectos de desempenho de cada modelo. Os estudos de caso 1 e 2 tiveram como foco as técnicas de otimização, mostrando as vantagens de se quantizar um modelo de ML para dispositivos com baixo recurso de memória. O terceiro estudo de caso trouxe uma aplicação real, mostrando a grande diferença em se ter o processamento no dispositivo de borda.

7.2 Trabalhos Futuros

Como trabalhos futuros dessa dissertação, sugere-se a realização de novos estudos voltados à analise do sistema proposto com outras aplicações, que possam envolver outros modelos de *machine learning*, bem com a utilização de outras redes de longo alcance e baixa potência. Explorar aplicações de reconhecimento padrões em imagens, bem como predição de eventos podem ser explorados também são caminhos promissores.

Aplicações que envolvam a necessidade de processamento e decisão em poucos milissegundos podem se aproveitar do sistema proposto. Por conta do tipo de aplicação utilizado para demonstrar as avaliações de modelos otimizados, recomenda-se também o

refinamento na determinação do *threshold*, que aqui foi simplificada por não ser o foco do trabalho. Foi possível verificar na seção de resultados que, caso esse parâmetro fosse melhor refinado, o resultado das matrizes confusão seriam significativamente superiores.

Neste trabalho foram apresentados estudos de caso onde foram avaliados técnicas de otimização de modelos de ML de forma individual. Pode-se, a partir dos resultados, realizar a utilização simultânea de otimizações, como por exemplo a quantização pós-treinamento e a técnica de poda. Essa sugestão agregaria dois fatores positivos observados nos estudos de caso: a redução da latência por inferência que a quantização proporciona e a redução do modelo compactado que a técnica de poda oferece.

REFERÊNCIAS

ABADI, M. *et al.* **TensorFlow**: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

AHMAD, U. *et al.* Analysis of Classification Techniques for Intrusion Detection. *In*: INTERNATIONAL CONFERENCE ON INNOVATIVE COMPUTING (ICIC), 2019., 2019. **Anais [...]** [S.l.: s.n.], 2019. p. 1–6.

ALAN CHIAO, R. A. **TensorFlow Model Optimization Toolkit — Weight Clustering API**. Disponível em: <https://blog.tensorflow.org/2020/08/tensorflow-model-optimization-toolkit-weight-clustering-api.html>. Acesso em: 05/03/2022.

ALLAN, A. **How to add LoRaWAN to Raspberry Pi Pico**. Disponível em: <https://www.raspberrypi.com/news/how-to-add-lorawan-to-raspberry-pi-pico/>. Acesso em: 31/05/2022.

ALONGI, F. *et al.* Tiny Neural Networks for Environmental Predictions: an integrated approach with miosix. *In*: IEEE INTERNATIONAL CONFERENCE ON SMART COMPUTING (SMARTCOMP), 2020., 2020. **Anais [...]** [S.l.: s.n.], 2020. p. 350–355.

ANALOG DEVICES. **AMP02 - Product Ooverview**. https://www.analog.com/en/products/amp02.html/>. Acesso em: 20/06/2022.

ANDRADE, P. *et al.* An Unsupervised TinyML Approach Applied for Pavement Anomalies Detection Under the Internet of Intelligent Vehicles. *In*: IEEE INTERNATIONAL WORKSHOP ON METROLOGY FOR INDUSTRY 4.0 IOT (METROIND4.0 IOT), 2021., 2021. **Anais [...]** [S.l.: s.n.], 2021. p. 642–647.

API, T. **TensorFlow Model Optimization Python API**. Disponível em: <https://www.tensorflow.org/model_optimization/api_docs/python/tfmot/>. Acesso em: 23/04/2022.

BANBURY, C. R. *et al.* Benchmarking TinyML Systems: challenges and direction. **CoRR**, [S.l.], v. abs/2003.04821, 2020.

BANBURY, C. *et al.* **MicroNets**: neural network architectures for deploying tinyml applications on commodity microcontrollers. [S.l.]: arXiv, 2020.

CHASE, J. The evolution of the internet of things. **Texas Instruments**, [S.l.], v. 1, n. 1388, p. 1–7, 2013.

CHEN, J.; RAN, X. Deep Learning With Edge Computing: a review. **Proceedings of the IEEE**, [S.l.], v. 107, n. 8, p. 1655–1674, 2019.

CISCO, V. Cisco visual networking index: forecast and trends, 2017–2022. **White paper**, [S.l.], v. 1, n. 1, 2018.

COOK, A. A.; MıSıRLı, G.; FAN, Z. Anomaly Detection for IoT Time-Series Data: a survey. **IEEE Internet of Things Journal**, [S.l.], v. 7, n. 7, p. 6481–6494, 2020.

DE PRADO, M. *et al.* Robustifying the Deployment of tinyML Models for Autonomous Mini-Vehicles. *In*: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2021., 2021. **Anais [...]** [S.l.: s.n.], 2021. p. 1–5.

DOYU, H.; MORABITO, R. **TinyML as-a-Service**: what is it and what does it mean for the iot edge? 2019.

FADLULLAH, Z. M. *et al.* Toward intelligent machine-to-machine communications in smart grid. **IEEE Communications Magazine**, [S.l.], v. 49, n. 4, p. 60–65, 2011.

GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, MA, USA: MIT Press, 2016.

HAN, S.; MAO, H.; DALLY, W. J. Deep Compression: compressing deep neural network with pruning, trained quantization and huffman coding. *In*: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, ICLR 2016, SAN JUAN, PUERTO RICO, MAY 2-4, 2016, CONFERENCE TRACK PROCEEDINGS, 4., 2016. **Anais [...]** [S.l.: s.n.], 2016.

HAN, S. *et al.* **ADBench**: anomaly detection benchmark. 2022.

J. LEE H. QIU, J. L.; SERVICES, R. T. **IMS, University of Cincinnati. "Bearing Data Set", NASA Ames Prognostics Data Repository**. Disponível em: <http://ti.arc.nasa.gov/project/prognostic-data-repository/>. Acesso em: 16/03/2022.

JACOB, B. *et al.* Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *In*: IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2018., 2018. **Proceedings [...]** [S.l.: s.n.], 2018. p. 2704–2713.

LINUXISE. **Gzip Command in Linux**. Disponível em: <https://linuxize.com/post/gzip-command-in-linux/>. Acesso em: 05/03/2022.

LITE, T. **Deploy Machine Learning Models on Mobile and Edge Devices**. Disponível em: <https://www.tensorflow.org/lite>. Acesso em: 20/03/2022.

LOUP GAILLY, J.; FOUNDATION, F. S. **gzip - Compress files with zip algorithm and 'compress' interface**. Disponível em: <https://opensource.apple.com/source/gnuzip/gnuzip-28/gzip/gzip.c.auto.html>. Acesso em: 05/06/2022.

LTD, R. P. **Raspberry Pi Pico C/C++ SDK - Libries and tools for C/C++ development on RP2040 microcontrollers**. Disponível em: <https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf/>. Acesso em: 31/05/2022.

LUO, T.; NAGARAJAN, S. G. Distributed Anomaly Detection Using Autoencoder Neural Networks in WSN for IoT. *In*: IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS (ICC), 2018., 2018. **Anais [...]** [S.l.: s.n.], 2018. p. 1–6.

MERENDA, M.; PORCARO, C.; IERO, D. Edge Machine Learning for AI-Enabled IoT Devices: a review. **Sensors**, [S.l.], v. 20, p. 2533, 04 2020.

ML, T. **Tiny ML Foundation**. Disponível em: <https://www.tinyml.org/>. Acesso em: 20/03/2022.

MOALLEMI, A. *et al.* Exploring Scalable, Distributed Real-Time Anomaly Detection for Bridge Health Monitoring. **IEEE Internet of Things Journal**, [S.l.], p. 1–1, 2022.

MULLIGAN, G. Foreword. In: HöLLER, J. *et al.* (Ed.). **From Machine-To-Machine to the Internet of Things**. Oxford: Academic Press, 2014. p. xiii–xiv.

NETA ZMORA, H. W.; RODGE, J. **TAchieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with NVIDIA TensorRT**. Disponível em: <https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/>. Acesso em: 12/03/2022.

PARMA, G. G.; MENEZES, B. R. D.; BRAGA, A. P. Neural networks learning with sliding mode control: the sliding mode backpropagation algorithm. **International Journal of Neural Systems**, [S.l.], v. 9, n. 03, p. 187–193, 1999.

PI, R. **Raspberry Pi Pico**. Disponível em: <https://www.raspberrypi.com/products/raspberry-pi-pico/>. Acesso em: 31/05/2022.

REN, H.; ANICIC, D.; RUNKLER, T. A. TinyOL: tinyml with online-learning on microcontrollers. *In*: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), 2021., 2021. **Anais [...]** [S.l.: s.n.], 2021. p. 1–8.

ROY, M. *et al.* A Stacked Autoencoder Neural Network based Automated Feature Extraction Method for Anomaly detection in On-line Condition Monitoring. *In*: IEEE SYMPOSIUM SERIES ON COMPUTATIONAL INTELLIGENCE (SSCI), 2018., 2018. **Anais [...]** [S.l.: s.n.], 2018. p. 1501–1507.

SANCHEZ-IBORRA, R.; SKARMETA, A. F. TinyML-Enabled Frugal Smart Objects: challenges and opportunities. **IEEE Circuits and Systems Magazine**, [S.l.], v. 20, n. 3, p. 4–18, 2020.

SARKER, V. K. *et al.* A Survey on LoRa for IoT: integrating edge computing. *In*: FOURTH INTERNATIONAL CONFERENCE ON FOG AND MOBILE EDGE COMPUTING (FMEC), 2019., 2019. **Anais [...]** [S.l.: s.n.], 2019. p. 295–300.

SEMTECH. **SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver**. Disponível em: <https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276/>. Acesso em: 15/03/2022.

SUDHARSAN, B. *et al.* OTA-TinyML: over the air deployment of tinyml models and execution on iot devices. **IEEE Internet Computing**, [S.l.], v. 26, n. 3, p. 69–78, 2022.

SURESH, V. M. *et al.* Powering the IoT through embedded machine learning and LoRa. *In*: IEEE 4TH WORLD FORUM ON INTERNET OF THINGS (WF-IOT), 2018., 2018. **Anais [...]** [S.l.: s.n.], 2018. p. 349–354.

SWAMY, S. N.; KOTA, S. R. An Empirical Study on System Level Aspects of Internet of Things (IoT). **IEEE Access**, [S.l.], v. 8, p. 188082–188134, 2020.

TOSHNIWAL, A.; MAHESH, K.; JAYASHREE, R. Overview of Anomaly Detection techniques in Machine Learning. *In*: FOURTH INTERNATIONAL CONFERENCE ON I-SMAC (IOT IN SOCIAL, MOBILE, ANALYTICS AND CLOUD) (I-SMAC), 2020., 2020. **Anais [...]** [S.l.: s.n.], 2020. p. 808–815.

VALADÃO, Y. d. N. *et al.* Wireless Sensor Network for Monitoring Reproductive Events in Cattle. *In*: INTERNATIONAL SYMPOSIUM ON INSTRUMENTATION SYSTEMS, CIRCUITS AND TRANSDUCERS (INSCIT), 2021., 2021. **Anais [...]** [S.l.: s.n.], 2021. p. 1–5.

WEIGERT, J. **xxd(1) - Linux man page**. Disponível em: <https://linux.die.net/man/1/xxd>. Acesso em: 25/05/2022.

XIONG, X. *et al.* Low power wide area machine-to-machine networks: key techniques and prototype. **IEEE Communications Magazine**, [S.l.], v. 53, n. 9, p. 64–71, 2015.

ZHU, M.; GUPTA, S. **To prune, or not to prune**: exploring the efficacy of pruning for model compression. [S.l.]: arXiv, 2017.

ZIV, Y.; GOLDBERGER, J.; Riklin Raviv, T. Stochastic weight pruning and the role of regularization in shaping network structure. **Neurocomputing**, [S.l.], v. 462, p. 555–567, 2021.