# Avaliação de Modelos Otimizados de *TinyML* para Detecção de Anomalias em IoT ⋆

**Leomar Mateus Radke** * **Max Feldman** * **Ivan Müller** *

* *Universidade Federal do Rio Grande do Sul (UFRGS)*
*Programa de Pós-Graduação em Engenharia Elétrica (PPGEE)*
*Porto Alegre, Rio Grande do Sul, Brasil*
*(e-mail: leomar@hotmail.com, max.feldman@ufrgs.br, ivan.muller@ufrgs.br).*

**Abstract:** The advancement of Internet of Things (IoT) applications in the context of low-power long-distance networks today is notorious. However, some weaknesses also appeared, such as the security of the transmitted data, 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 6x drop in power consumption in the edge anomaly detection.

**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 também apareceram, como a segurança dos dados trafegados, largura de banda e autonomia de bateria dos dispositivos. Este trabalho apresenta uma avaliação de modelos otimizados de *Tiny Machine Learning* (*TinyML*). São avaliados os benefícios de se ter em um dispositivo sensor um algoritmo otimizado, onde a inferência dos dados é realizada localmente. Será avaliada a performance de cada uma das técnicas, 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 foi uma redução de quase 6 vezes no consumo de energia na proposta de detecção de anomalia na borda.

*Keywords:* Optimization, *TinyML*, Anomaly Detection, Low-Power Wide Area Network.
*Palavras-chaves:* Otimização, *TinyML*, Detecção de Anomalias, Redes de Longa Distância de Baixa Potência.

## 1. INTRODUÇÃO

O impacto econômico potencial da *Internet of Things* (IoT) está projetado para atingir até 11,1 trilhões de dólares anualmente até 2025. A maioria dos dados de IoT coletados hoje não são usados e, os dados utilizados, não são totalmente explorados. Os casos de uso mais corriqueiros são para controle em tempo real ou detecção de anomalias em sistemas de automação de fabricação (Manyika et al., 2015).

Dentro da IoT, uma das técnicas mais difundidos está a coleta de dados em dispositivos sensores para posterior envio a uma solução centralizada, que irá processar, transformar e realizar uma determinada inferência. Esse processo, dependendo da aplicação, pode apresentar fragilidades ou ineficiência da operação. Ter todos os dados sensíveis de uma aplicação trafegando até um nó concentrador, limitações de banda e bateria dos dispositivos finais da rede – como pode ser visto em Redes de Longa Distância de Baixa Potência, do inglês *Low Power Wide Area Network* (LPWAN), 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 de dados desnecessários, reduzindo consumo de bateria na transmissão e trazendo maior segurança, visto que os dados da aplicação podem não ser transmitidos.

Evidentemente que, 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 rodar com pouca memória RAM. Dentro do conceito de IA, o *Machine Learning* (ML) é o bloco de construção essencial para a IoT. Devido à ampla implantação e ao consumo mínimo de energia dos dispositivos IoT, pesquisadores prestam cada vez mais atenção para fornecer funcionalidades de ML na borda (Ren et al., 2021).

Para preencher a lacuna entre microcontroladores e ML, especialmente aprendizado profundo, o *TinyML* dedica-se a oferecer soluções baseadas em Redes Neurais Artificiais

⋆ O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

(RNA) na borda. Resultados notáveis foram produzidos recentemente, como o reconhecimento de voz (Sanchez-Iborra and Skarmeta, 2020) e previsões de pressão atmosférica (Alongi et al., 2020). As principais vantagens da utilização dessa abordagem 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: Uma rede neural consome muita energia, porém, a transmissão dos dados para a nuvem precisa de uma ordem de magnitude maior de energia.

O objetivo geral do presente trabalho será a elaboração de um sistema de processamento para detecção de anomalia na borda em uma LPWAN, avaliando modelos otimizados para hardwares reduzidos. Os objetivos específicos do trabalho são:

- Comparar o desempenho de consumo de energia do dispositivo da borda. Primeiramente, em um cenário de transmissão completa dos dados, para que a inferência seja realizada remotamente – em um sistema tipico de processamento *cloud*. Na sequência, aplicar a solução para que a inferência seja realizada na borda, transmitindo apenas em caso de detecção de anomalia.
- Realizar comparações entre o modelo completo de RNA e os modelos otimizados, apresentado métricas de precisão, sensibilidade e acurácia, além de apresentar o tamanho de cada arquivo em quantidade de bytes.

## 2. TRABALHOS RELACIONADOS

Existem várias aplicações de IoT em que os nós de borda são simples sensores que coletam dados e os transferem para servidores por meio de *gateways*. Porém, trabalhos recentes apresentam propostas de alteração deste cenário.

Em Merenda et al. (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. Como prova do poder que essa nova abordagem pode trazer, os autores apresentam um simples ensaio, onde o dataset MNIST é treinado por meio do *framework TensorFlow*, otimizado pela técnica de poda de pesos e posteriormente testado em um dispositivo embarcado, alcançando acurácia de 100% na detecção de alguns dígitos.

Aos poucos, aplicações também vão ganhando importância dentro do conceito de *TinyML*. É proposto em Andrade et al. (2021) a de detecção de anomalias – buracos, solavancos, obstáculos em estradas, sendo utilizado um microcontrolador e um acelerômetro embarcado em um veículo.

Em Suresh et al. (2018), os autores demonstraram em seu trabalho como o processamento na borda pode prolongar a bateria de dispositivos. Foi avaliado a saúde de animais em uma fazenda, onde o microcontrolador possui um algoritmo treinado para classificar determinado comportamento. Eles realizaram a comparação de inferência local e em um *cloud* utilizando uma rede *LoRa*. O resultado disso foi um aumento de autonomia de seu dispositivo de 13 dias para 39 dias. Essa mesma abordagem, de uso de uma rede de baixa potência de longo alcance foi utilizada em Moallemi et al. (2022), onde os autores construíram uma aplicação de detecção de anomalias e integridade de uma ponte. Três abordagens de algoritmos foram utilizadas aqui: Análise de Componente Principal, *Autoencoder* Totalmente Conectado e *Autoencoder* Convolucional. Os autores conseguiram reduzir o consumo em aproximadamente 5x devido ao baixo tráfego de dados na rede.

Entretanto, em ambos os trabalhos, não foi abordado o tema de otimização dos modelos criado.

A Tabela 1 apresenta uma visão geral dos trabalhos avaliados. O compilado de características e principais contribuições que o presente trabalho pretende abordar estão na última linha da tabela.

Até onde foi possível pesquisar, não há um trabalho que avalie modelos otimizados de *Machine Learning* para detecção de anomalias em aplicações com dispositivos com poucos recursos hardware em redes de baixa potência.

## 3. FUNDAMENTAÇÃO TEÓRICA

### *3.1 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 máquina a máquina. Os dispositivos IoT empregam uma ampla gama de protocolos de rede, aplicativos e domínios de rede (Mulligan, 2014). A IoT envolve o desenvolvimento de uma rede invisível e inteligente que pode ser controlada e programada. Os dispositivos utilizados nesta rede utilizam a tecnologia embarcada, que permite comunicar direta ou indiretamente utilizando a infraestrutura da Internet (Chase, 2013). Muitas aplicações de IoT 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, etc. Com a utilização da IoT, uma alta quantidade de informações, das mais diversas aplicações, estão sendo criadas. No entanto, existem alguns pontos de atenção, como a segurança da informação. Esse 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.

### *3.2 Redes de Longa Distância de Baixa Potência*

As redes de longa distância de baixa potência, *Low-Power Wide-Area Network* (LPWAN), representam um novo paradigma de comunicação, que complementa as tecnologias

Tabela 1. Relação de trabalhos e suas contribuições.

| | Avaliação de Aplicação | | Edge Computing | Otimização de Modelos | | | |
|---|---|---|---|---|---|---|---|
| | Consumo de Energia | Detecção de Anomalia | TinyML em LPWAN | Quantização Pós-Treino | Treinamento Consciente de Quantização | *Pruning* | Agrupamento de Pesos |
| Suresh et al. (2018) | x | x | x | - | - | - | - |
| Moallemi et al. (2022) | x | - | x | - | - | - | - |
| Andrade et al. (2021) | - | x | - | - | - | - | - |
| Merenda et al. (2020) | - | - | x | - | - | x | - |
| Este Trabalho | x | x | x | x | x | x | x |

celulares tradicionais e sem fio de curto alcance, como Zigbee/IEEE 802.15.4 em redes de sensores sem fio, Bluetooth, RFID, afim de atender a diversos requisitos de aplicativos de IoT (Fadlullah et al., 2011).

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 de área ampla de baixa potência 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 espectro de dispersão de sequência direta foram propostos para a camada física de sistemas LPWAN *Machine-to-Machine* (M2M) graças ao seu excelente desempenho de cobertura.

Enquanto isso, para possibilitar baixo consumo de energia, a topologia em estrela e o método de acesso aleatório podem ser empregados na camada *Media Access Control*. Uma característica fundamental dessas técnicas LPWAN é que elas fornecem uma compensação entre a taxa de dados, vida útil da bateria e custos de implantação. Essa compensação é aceitável para a maioria da aplicações M2M, que não exigem altas taxas de dados e baixa latência (Xiong et al., 2015).

### *3.3 TinyML*

*TinyML* é um conceito promissor que lida com a execução de modelos de ML otimizados em 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, o *TinyML* pode se tornar uma ferramenta valiosa para aprimorar os recursos de processamento, especialmente porque o processamento de dados e os serviços de ML no dispositivo ampliam a capacidade do mesmo.

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 de baixo custo, o *TinyML* atraiu muita atenção da academia e da indústria. Especialistas em pesquisa de grandes empresas de tecnologia e acadêmicos colaboraram 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* (Tiny ML, 2019).

### *3.4 Otimização de Modelos de Machine Learning*

Um importante aspecto dentro do *TinyML* está a preocupação 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.

- Quantização Pós-Treino - é aquele em que se representam os modelos com menor precisão, como inteiros de 8 bits em oposição a float de 32 bits.
- Treinamento Consciente de Quantização - aplica a mesma quantização do item acima no treinamento durante o processo *forward* da rede, enquanto o processo *backward* permanece o mesmo, com valores em float.
- *Pruning* - A poda de um modelo de ML consiste em remover (definir como 0 permanentemente) certos pesos. Normalmente os pesos que são 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. É fornecida uma esparsidade inicial, uma esparsidade final, uma etapa para iniciar a poda, uma etapa para terminá-la e, finalmente, o expoente do decaimento polinomial, como visto na Equação 1,
$$S = (S_f - S_0)\frac{t - t_0}{t_e - t_0}^{\alpha} \quad (1)$$
onde S é a esparsidade, $S_f$ é a esparsidade final, $S_0$ é a esparsidade inicial, $t$ é a etapa de tempo atual, $t_e$ é a etapa final, $t_0$ é a etapa inicial e $\alpha$ é o expoente, o valor padrão é 3.
- 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*.

### *3.5 RNA 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 and Nagarajan, 2018). Uma típica RNA *autoencoder* é mostrada na Figura 1.

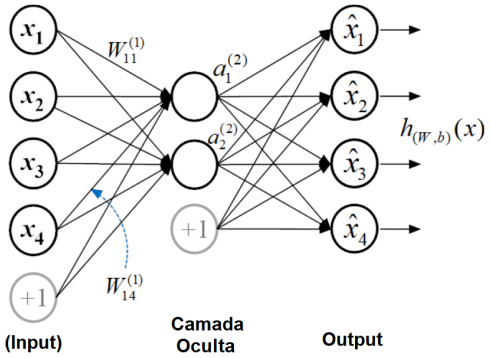


Figura 1. Exemplo de uma RNA *autoencoder* (Luo and Nagarajan, 2018).

(1) Uma camada de entrada: um vetor de dimensão M que representa os sinais de entrada, denotado por: $\mathbf{x} = (x_1, x_2, ..., x_M)$
(2) Uma 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: $\mathbf{y} = (y_1, y_2, ..., y_M)$ 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-ésimo 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 et al., 2020).

## 4. SISTEMA PROPOSTO

Essa seção apresenta dois aspectos importantes do trabalho. O primeiro é a definição da arquitetura do sistema proposto para avaliar as técnicas de otimização. Em 4.1, são definidos e explicados os passos para se criar uma solução de *TinyML* e avaliação de um cenário de aplicação em uma LPWAN. O outro aspecto importante, apresentado em 4.2, é a avaliação da performance de cada técnica de otimização apresentada. Para tal, são colocadas também as características da RNA utilizada, definições no modo de treinamento do modelo e as avaliação desse treinamento.

### *4.1 Arquitetura e sua avaliação*

Para realizar a avaliação dos modelos otimizados, bem apresentar os benefícios de se realizar inferências em dispositivos de borda, é necessário estabelecer uma arquitetura. Conforme apresentado na Figura 2, a arquitetura do sistema possui diversas etapas, logo devem ser estabelecidos critérios para avaliação. Primeiramente é importante definir onde cada processo é 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 et al., 1999).

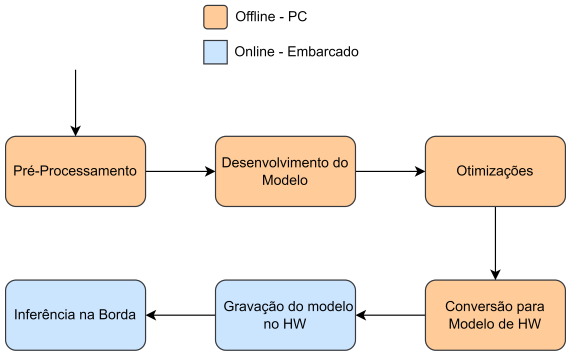


Figura 2. Etapas da construção de uma solução *TinyML*.

Todos esses processos são realizados em linguagem Python. Como pilar central do desenvolvimento, são exploradas as funcionalidades e ferramentas disponíveis no *framework TensorFlow* (TensorFlow, 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 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 em 3.4 são aplicadas neste momento.
(4) Conversão para Modelo de Hardware: por fim, para cada modelo otimizado, um arquivo é gerado. Posteriormente, cada arquivo é anexado ao software embarcado, gerando um binário por otimização.

Já os blocos da Figura 2 em azul, legendados como *Online - Embarcado*, são tratados no dispositivo de borda. O hardware adotado para este trabalho é uma Raspberry Pi Pico. 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, robusto *Software Development Kit* (SDK) e compatibilidade com a ferramenta de conversão de modelos *TensorFlow*, o *TensorFlow Lite*.

Estabelecido o fluxo dos processos necessários para a construção de uma solução *TinyML*, faz-se necessário a criação de um cenário onde se possa aplicar a inferência na borda. Assim, será possível observar quais os benefícios obtidos com a arquitetura proposta. O formato da aplicação será a de detecção de anomalias.

Para a criação dessa aplicação, uma LPWAN do tipo *LoRa* será utilizada. Uma rede será estabelecida entre o dispositivo da borda e um dispositivo coletor de informações - um Arduino Nano, neste caso. Ambos os dispositivos serão dotados de um módulo *LoRa*. O módulo SX1276 é um transceptor, dispositivo responsável pela comunicação de espectro disperso, longo alcance, alta imunidade a interferências e baixo consumo. Apenas a camada física do protocolo *LoRa* é utilizada. Neste trabalho é utilizado um fator de espalhamento 8 durante a execução de todos os testes descritos.

A avaliação desse cenário, ilustrada na Figura 3, terá dois modos de operação. O primeiro, com a caixa e seta em azul, é quando o dispositivo de borda envia o conteúdo completo da amostra, ou seja, faz uma transferência total do conteúdo de dados disponíveis. O dispositivo coletor está conectado a um computador, que irá realizar a inferência dos dados.

O segundo modo de operação será quando o dispositivo de borda irá inferir a amostra e enviar para o coletor apenas quando existir algum tipo de anomalia. Não será enviado o conteúdo, apenas uma notificação de detecção.

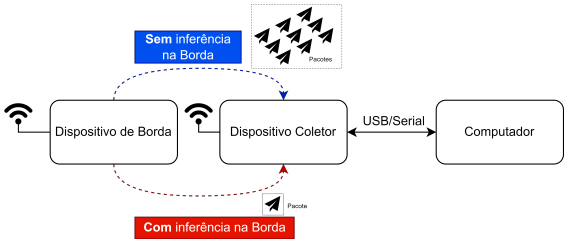


Figura 3. Cenário de aplicação para avaliação da arquitetura proposta.

O fator preponderante considerado para comparação dos dois modos de operação será o consumo de energia. A medição desse consumo será realizada por um terceiro dispositivo, um microcontrolador, que irá realizar computar os valores de tensão, por meio de um conversor analógico-digital, de um resistor *shunt* colocado na entrada da alimentação do dispositivo de borda.

### *4.2 Avaliação das Técnicas de Otimização*

A metodologia de teste da arquitetura do sistema proposto possui uma etapa não mapeada na Figura 2, a coleta de dados. Nesse trabalho, optou-se pela utilização de um dataset já estabelecido, disponível em J. Lee, H. Qiu, G. Yu, J. Lin and Rexnord Technical Services (2007). Este dataset é composto por amostras de 4 acelerômetros, onde cada um deles está amostrando o comportamento de rolamentos (*bearings*) acoplados a um motor. Os dados foram coletados a cada 10 minutos em um período de 30 dias. Um visão do comportamento das amostras pode ser vista na Figura 4. O eixo $y$ representa os valores coletados pelos acelerômetros.

É possível notar que os rolamentos apresentavam 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

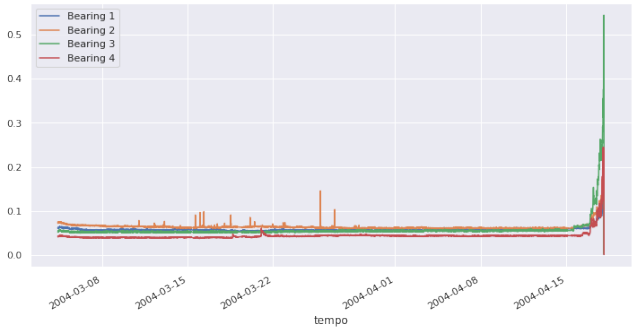


Figura 4. Visão geral das amostras do dataset utilizado.

4 dias antes da falha acontecer de fato. Para avaliar esse comportamento, as amostras de treinamento do modelo serão as coletadas antes do dia 2004-04-11 e, consequentemente, as demais serão as de teste. Essa escolha se deve ao fato da utilização de *Autoencoder*, onde a rede busca reconstruir a entrada com o menor erro possível. Dessa forma, escolhendo apenas amostras cujo o comportamento é padrão, quando as amostras que possuem alguma anomalia passarem pelo *Autoencoder*, o erro de reconstrução será significativamente maior.

A topologia da rede utilizada nesse trabalho está descrita na Tabela 2. A avaliação do treinamento utiliza o *Mean Squared Error* (MSE). Todas as funções de ativação são do tipo *Rectified Linear Unit* (ReLU), ou unidade linear retificada e o número de épocas de treinamento foi estipulada em 100.

Tabela 2. Número de nós por camada.

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

Após o treinamento, 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 Average Error* (MAE). Como apresentado na Figura 5, a distribuição do MAE absoluto nas amostras avaliadas tem como maior valor 0.25. Esse é um valor importante, pois será ele o *threshold* adotado para avaliar os dados de teste. Caso a amostra tenha um MAE superior a 0.25, será considerada uma amostra com anomalia, caso contrário, será considerada normal.

Após essa definição, aplica-se o modelo treinado nos dados de teste. Na Figura 6 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, serão adotadas como comportamento normal 768 amostras e anormais 357 amostras. Esse procedimento de rotular as amostras é adotado para verificar se os modelos otimizados terão o mesmo resultado do modelo não otimizado.

Até aqui, todo o desenvolvimento tratou do modelo *TensorFlow* não otimizado. Com o modelo base criado, são aplicadas as técnicas de otimização. Para quantização pós-treino, é preciso calibrar ou estimar o intervalo de todos os nós de ponto flutuante no modelo. Ao contrário de nós constantes, como pesos e vieses, tensores variáveis, como

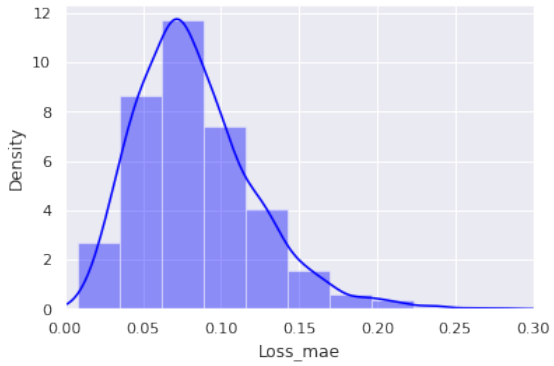


Figura 5. Distribuição do MAE da reconstrução dos dados de entrada.

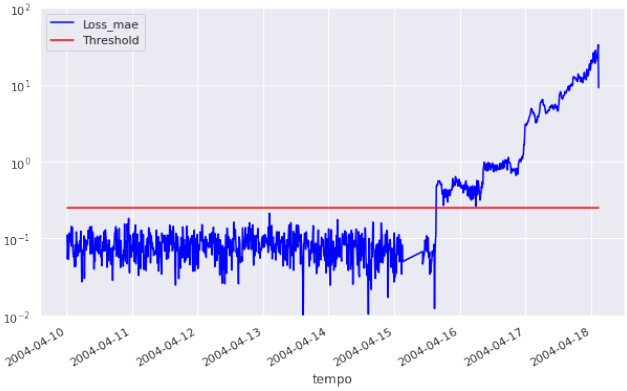


Figura 6. MAE dos dados de teste.

entrada de modelo, ativações (saídas de camadas intermediárias) e saída de modelo, não podem ser calibrados a menos que seja executado 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 (cerca de 100-500 amostras) dos dados de treinamento ou validação (TensorFlow, 2015).

No treinamento consciente de quantização, poda e agrupamento de pesos, o mesmo número de épocas, camadas e 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.

Após criar todos os modelos otimizados, pode-se então converter esses arquivos para modelos *TensorFlow Lite*, que possibilitam a portabilidade para o embarcado utilizado neste trabalho. Uma aplicação em C/C++ é gerada para receber os modelos. Dentro dessa aplicação, também serão carregados os dados de teste – 1125 amostras.

## 5. RESULTADOS

Os resultados são apresentados com duas visões distintas. Na Seção 5.1, são apresentadas as comparações entre os modelos otimizados. Já na Seção 5.2 apresenta as métricas de desempenho da arquitetura proposta em 4.1.

### *5.1 Modelos Otimizados*

A eficiência da inferência é uma preocupação crítica ao implantar modelos de ML devido à latência, à utilização de memória e, em muitos casos, ao consumo de energia. Especialmente em dispositivos de borda, os recursos são ainda mais restritos, e o tamanho do modelo e a eficiência da computação são uma grande preocupação. Dependendo da aplicação, será necessário fazer uma compensação entre a complexidade do modelo e o tamanho do mesmo. Em casos onde não seja exigida uma grande acurácia, pode ser mais indicado usar um modelo menor porque eles não apenas usam menos espaço em disco e memória, mas também são geralmente mais rápidos.

A Tabela 3 apresenta os resultados encontrados para cada uma das técnicas de otimização avaliadas. Para melhor visualização, atribuiu-se a seguinte legenda para cada uma delas:

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

Tabela 3. Tamanho final do modelo para diferentes técnicas 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 explicado pelo fato de que nessa técnica os pesos menos significantes 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, onde o mesmo modelo é compactado por programa *gzip* (Linuxise, 2019). 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 et al., 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*.

Porém, de nada adiantaria reduzir o modelo A1 para as demais variantes sem avaliar a qualidade que eles entregam, baseado na matriz confusão (2) dos dados de teste. Os casos Verdadeiro Positivo (VP), Falso Positivo (FP), Falso Negativo (FN) e Verdadeiro Negativo (VN) fazem a composição da matriz.

$$M_C = \begin{bmatrix} VP & FP \\ FN & VN \end{bmatrix} \tag{2}$$

A partir disso, calcula-se a precisão (4) – onde CP é o somatório de condições positivas (VP e FP), a sensibilidade (3) – onde CPP é o somatório das condições positivas previstas (VP e FN), para obtenção da $F1_{Score}$ (Ahmad et al., 2019).

$$Recall = \frac{VP}{CP} \tag{3}$$

$$Precision = \frac{VP}{CPP} \tag{4}$$

$$F1_{Score} = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \tag{5}$$

É calculada também a *accuracy* (6), onde TA é o somatório de todas as amostras avaliadas.

$$Accuracy = \frac{(VP + VN)}{(TA)} \tag{6}$$

Na Tabela 4 são mostrados os valores encontrados. Em A1, o modelo *TensorFlow Lite* obteve exatamente o mesmo resultado do modelo *TensorFlow* desenvolvido e testado em um computador, que foi ilustrado na Figura 6. Esse resultado por si só já é de grande entusiasmo, 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 4. 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. Novamente, por ser a técnica mais simplificada dentre as apresentadas. Os demais modelos apresentaram pouca diferença. Em todos os modelos, houve a mesma detecção do primeiro VP - amostra de número 769.

### *5.2 Arquitetura*

Nessa avaliação, será apresentado apenas o resultado para modelo não otimizado A1. Visto que não estão sendo avaliadas as características do modelo, mas sim uma solução de inferência na borda, não foram constatadas diferenças no consumo energético do dispositivo quando estavam sendo executados os demais modelos.

Na Tabela 5 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, 1125. No modo com inferência no embarcado da borda, todas as 357 amostras classificadas como anômalos são envidadas ao coletor.

Tabela 5. Comparação entre a operação com e sem inferência na borda.

| | Sem inferência | Com inferência |
|---|---|---|
| Quantidade de envios | 1125 | 357 |
| Tamanho pacote (bytes) | 28 | 14 |
| Tempo envio (s) | 0,123 | 0,080 |
| Energia por envio(J) | 0,030 | 0,018 |
| Energia total (J) | 33,665 | 6,426 |

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 informar o rolamento com problema. Na Figura 7 a concatenação dos pacotes é ilustrada. 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.

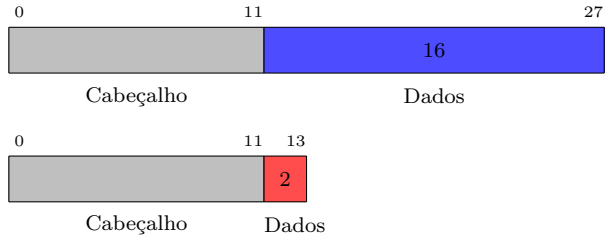


Figura 7. Estrutura de um pacote enviado em ambos as soluções propostas.

Esses dois fatores combinados, resultaram em uma redução de quase 6 vezes no consumo, resultado muito similar ao observado em Moallemi et al. (2022). Um possível *trade-off* causado por executar a inferência no embarcado foi descartado, pois não foi percebido um consumo de energia significativamente diferente quando executando a inferência na borda. Em aplicações onde haja 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.

## 6. CONCLUSÃO

Este trabalho propôs uma avaliação de modelos otimizados de *TinyML* para detecção de anomalias. Foi possível verificar os benefícios diretos e comprovar a viabilidade de uma arquitetura para desenvolvimento de soluções de ML para dispositivos com poucos recursos de hardware. Todas as técnicas de otimização tiveram sucesso na detecção do momento definido como ponto de inflexão entre amostras

normais e com anomalia. A proposta que avaliou o consumo energético da solução demonstrou o quanto vantajosa pode ser a inferência na borda. Houve uma redução de quase 6 vezes no consumo de energia, sem contar o fato de não expor nenhum dado relevante do conjunto de dados. Na situação proposta, o tempo para inferência não era algo crítico, entretanto, em outros contextos, isso pode ser levado em conta também. Outro beneficio explícito está na redução que os técnicas fornecem, principalmente quando observado os arquivos compactados. Esse resultado pode trazer benefícios em futuras implementações, como a atualização dos modelos por meio do envio de novos vetores de pesos ao dispositivo de borda.

## REFERÊNCIAS

Ahmad, U., Asim, H., Hassan, M.T., and Naseer, S. (2019). Analysis of classification techniques for intrusion detection. In *2019 International Conference on Innovative Computing (ICIC)*, 1–6. doi:10.1109/ICIC48496.2019.8966675.

Alongi, F., Ghielmetti, N., Pau, D., Terraneo, F., and Fornaciari, W. (2020). Tiny neural networks for environmental predictions: An integrated approach with miosix. In *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, 350–355. doi:10.1109/SMARTCOMP50058.2020.00076.

Andrade, P., Silva, I., Signoretti, G., Silva, M., Dias, J., Marques, L., and Costa, D.G. (2021). An unsupervised tinyml approach applied for pavement anomalies detection under the internet of intelligent vehicles. In *2021 IEEE International Workshop on Metrology for Industry 4.0 IoT (MetroInd4.0 IoT)*, 642–647. doi:10.1109/MetroInd4.0IoT51437.2021.9488546.

Banbury, C.R., Reddi, V.J., Lam, M., Fu, W., Fazel, A., Holleman, J., Huang, X., Hurtado, R., Kanter, D., Lokhmotov, A., Patterson, D.A., Pau, D., Seo, J., Sieracki, J., Thakker, U., Verhelst, M., and Yadav, P. (2020). Benchmarking tinyml systems: Challenges and direction. *CoRR*, abs/2003.04821. URL `https://arxiv.org/abs/2003.04821`.

Chase, J. (2013). The evolution of the internet of things. *Texas Instruments*, 1(1388), 1–7.

Cook, A.A., Mısırlı, G., and Fan, Z. (2020). Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7), 6481–6494. doi:10.1109/JIOT.2019.2958185.

Fadlullah, Z.M., Fouda, M.M., Kato, N., Takeuchi, A., Iwasaki, N., and Nozaki, Y. (2011). Toward intelligent machine-to-machine communications in smart grid. *IEEE Communications Magazine*, 49(4), 60–65. doi:10.1109/MCOM.2011.5741147.

Han, S., Mao, H., and Dally, W.J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Y. Bengio and Y. LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. URL `http://arxiv.org/abs/1510.00149`.

J. Lee, H. Qiu, G. Yu, J. Lin and Rexnord Technical Services (2007). 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.

Linuxise (2019). Gzip Command in Linux. Disponível em: <https://linuxize.com/post/gzip-command-in-linux/>. Acesso em: 05/03/2022.

Luo, T. and Nagarajan, S.G. (2018). Distributed anomaly detection using autoencoder neural networks in wsn for iot. In *2018 IEEE International Conference on Communications (ICC)*, 1–6. doi:10.1109/ICC.2018.8422402.

Manyika, J., Chui, M., Bisson, P., Woetzel, J., Dobbs, R., Bughin, J., and Aharon, D. (2015). *The Internet of Things: Mapping the value beyond the hype*, volume 24. McKinsey Global Institute New York, NY, USA.

Merenda, M., Porcaro, C., and Iero, D. (2020). Edge machine learning for ai-enabled iot devices: A review. *Sensors*, 20, 2533. doi:10.3390/s20092533.

Moallemi, A., Burrello, A., Brunelli, D., and Benini, L. (2022). Exploring scalable, distributed real-time anomaly detection for bridge health monitoring. *IEEE Internet of Things Journal*, 1–1. doi:10.1109/JIOT.2022.3157532.

Mulligan, G. (2014). Foreword. In J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle (eds.), *From Machine-To-Machine to the Internet of Things*, xiii–xiv. Academic Press, Oxford. doi:https://doi.org/10.1016/B978-0-12-407684-6.00024-3. URL `https://www.sciencedirect.com/science/article/pii/B9780124076846000243`.

Parma, G.G., MENEZES, B.R.D., and Braga, A.P. (1999). Neural networks learning with sliding mode control: the sliding mode backpropagation algorithm. *International Journal of Neural Systems*, 9(03), 187–193.

Ren, H., Anicic, D., and Runkler, T.A. (2021). Tinyol: Tinyml with online-learning on microcontrollers. In *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–8. doi:10.1109/IJCNN52387.2021.9533927.

Sanchez-Iborra, R. and Skarmeta, A.F. (2020). Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3), 4–18. doi:10.1109/MCAS.2020.3005467.

Sarker, V.K., Queralta, J.P., Gia, T.N., Tenhunen, H., and Westerlund, T. (2019). A survey on lora for iot: Integrating edge computing. In *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 295–300. doi:10.1109/FMEC.2019.8795313.

Suresh, V.M., Sidhu, R., Karkare, P., Patil, A., Lei, Z., and Basu, A. (2018). Powering the iot through embedded machine learning and lora. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, 349–354. doi:10.1109/WF-IoT.2018.8355177.

TensorFlow (2015). TensorFlow model optimization. Disponível em: <https://www.tensorflow.org/>. Acesso em: 20/03/2022.

Tiny ML (2019). Tiny ML Foundation. Disponível em: <https://www.tinyml.org/>. Acesso em: 20/03/2022.

Xiong, X., Zheng, K., Xu, R., Xiang, W., and Chatzimisios, P. (2015). Low power wide area machine-to-machine networks: key techniques and prototype. *IEEE Communications Magazine*, 53(9), 64–71. doi:10.1109/MCOM.2015.7263374.