# Avaliação do Consumo de Energia para o Treinamento de Aprendizado de Máquina utilizando *Single-board computers* baseadas em ARM

**Felipe Bernardo, André Yokoyama, Bruno Schulze, Mariza Ferro**

[1]Laboratório Nacional de Computação Científica (LNCC)
Getúlio Vargas, 333, Quitandinha – Petrópolis – Rio de Janeiro

{felipeb,andre,schulze,mariza}@lncc.br

***Abstract.*** *In this work, the use of ARM-based single-board computers is evaluated for training Machine Learning (ML) algorithms. For this, an experimental setup was developed, training the algorithm XGBoost with 36 hyperparameter configurations in four different architectures. Furthermore, its efficiency (energy consumption, acquisition cost and execution time) was compared with the main architectures used for the training of ML (x86 and GPU). The results show that this type of architecture can become a viable and greener alternative, not only for inference but also for the training phase of these algorithms.*

***Resumo.*** *Neste trabalho é avaliado o uso de placas single-board computers baseadas em ARM para o treinamento de algoritmos de Aprendizado de Máquina (AM). Foi desenvolvido um conjunto experimental treinando o algoritmo XGBoost com 36 configurações de hiperparâmetros em quatro arquiteturas diferentes. Além disso, foi comparado a sua eficiência (consumo energético, custo de aquisição e tempo de execução) com as principais arquiteturas usadas no treinamento de algoritmos de AM (x86 e GPU). Os resultados mostram que este tipo de arquitetura pode se tornar uma alternativa viável e mais verde, não apenas para a inferência, mas também para a fase de treinamento desses algoritmos.*

## 1. Introdução

A Inteligência Artificial é vista como uma das forças mais transformadoras do nosso tempo, sendo considerada fundamental para fazer face a muitos dos grandes desafios que o mundo enfrenta em áreas como a saúde e o bem-estar e nas alterações climáticas, expressos como Objetivos de Desenvolvimento Sustentável (ODS) estabelecido pela Organização das Nações Unidas (ONU) [1]. No Brasil a IA já vive uma intensa profusão de iniciativas, as quais são vistas nas universidades e centros de pesquisa, na saúde, no setor financeiro, no agronegócio, na indústria e no setor público e governamental. Na última década foram realizados grandes progressos, principalmente na sub-área da IA denominada pelo Aprendizado de Máquina (AM). Esse progresso é resultado da convergência entre a atual disponibilidade de enormes quantidades de dados digitais e de arquiteturas de Computação de Alto Desempenho (HPC [2]), permitindo assim o desenvolvimento e o treinamento de algoritmos de AM de última geração.

[1]Sustainable Development Goals - https://www.un.org/sustainabledevelopment/
[2]Do original, em inglês, *High Performance Computing.*

Embora as aplicações de IA, incluindo o AM, tenham essa capacidade de gerar enormes benefícios para os indivíduos e para a sociedade, ela também dá origem a certas implicações ambientais que devem ser devidamente pensadas e geridas. Alguns estudos (Vinuesa et al. 2020; UNESCO 2021) vêm apontando que apesar da IA poder contribuir positivamente na solução da maioria dos ODS ela também pode impactar negativamente nas metas relacionadas às questões climáticas e ambientais. A principal razão para isso é que alguns modelos de AM podem ser muito custosos para serem treinados, tanto em termos computacionais como na demanda de energia elétrica, principalmente quando utilizando ambientes de HPC para o seu treinamento. Ambientes de HPC consomem muita energia para manter o seu funcionamento, incluindo núcleos de processamento e sistemas de refrigeração. É previsto que até 2030 metade do consumo de energia elétrica mundial será atribuído a instalações de computação e que aproximadamente 10% da energia elétrica global já é consumida com a mineração de dados (UNESCO 2021), o que está diretamente ligada ao uso de algoritmos de AM. O consumo de energia gera elevado custo monetário e também é responsável pela emissão de gases de efeito estufa, dentre os quais o dióxido de carbono ($CO_2$) é o mais expressivo.

Uma possível alternativa para reduzir o impacto que os algoritmos de AM causam na emissão de $CO_2$, é usar arquiteturas com baixo consumo energético, tais como ARM e RISC V. A arquitetura ARM é usada atualmente em *smartphones* e sistemas embarcados, os quais precisam de boa capacidade de processamento, porém com baixo consumo de energia, especialmente para a manutenção das baterias desses equipamentos. Porém, pouco se sabe sobre a viabilidade do uso de uma placa *single-board computers*, como Nvidia Jetson, ou Raspberry Pi para o treinamento de um algoritmo de AM. O que é feito tradicionalmente é treinar o algoritmo em uma arquitetura, por exemplo, do tipo GPGPU e x86 e com o modelo gerado, implementar apenas inferência em uma placa *single-board*.

Assim, o principal objetivo deste trabalho é avaliar a viabilidade do uso de placas *single-board computers* baseadas em ARM para o treinamento de algoritmos de AM. Além disso, comparar a sua eficiência (consumo energético, custo de aquisição e tempo de execução) com as principais arquiteturas tradicionalmente usadas para o treinamento de algoritmos de AM (x86 e GPU). Para isso foi desenvolvido um conjunto experimental treinando o algoritmo de AM XGBoost (Chen et al. 2015) com 36 configurações de hiperparâmetros, em quatro arquiteturas diferentes. Este algoritmo foi escolhido para avaliação por ser um modelo baseado em *ensemble* de árvores de decisão que vêm se mostrando muito eficiente para a solução de problemas com grandes volumes de dados estruturados, em diferentes domínios de aplicação. Além da boa precisão esta ainda é uma abordagem de AM explicável. Essas qualidades fazem dele o terceiro algoritmo de AM mais utilizado pela comunidade científica mundial, segundo o relatório (Kaggle 2020).

Este trabalho está organizado da seguinte forma: Na Seção 2 é descrito o algoritmo XGBoost e trabalhos relacionados, Seção 3 é apresentada a metodologia adotada para a execução dos experimentos. Na Seção 4 são apresentados os experimentos e os resultados obtidos. Na Seção 5 são apresentadas as considerações finais e os trabalhos futuros.

## 2. Fundamentação Teórica

Os algoritmos de AM supervisionado baseados em Árvores de Decisão (AD) estão entre os mais utilizados pela comunidade de ciência de dados. Segundo o re-

latório (Kaggle 2020), as AD e ensembles baseados em AD ocupam o segundo (Floresta Randômica) e terceiro lugar (*eXtreme Gradient Boosting* - XGBoost) da lista. Neste cenário, o algoritmo XGBoost (Chen et al. 2015) vem se destacando por sua capacidade preditiva, vencendo a maioria dos desafios do Kaggle nos últimos anos e também pelo seu desempenho, mesmo sem o uso de ambientes HPC (Silva et al. 2021).

O XGBoost é um modelo de *ensemble* do tipo Boosting. Ou seja, se constrói sucessivas hipóteses (AD), de tal modo que exemplos classificados incorretamente por hipóteses anteriores sejam melhor classificados nas hipóteses seguintes. O funcionamento do XGBoost é dado pela Equação 1, onde $K$ é o número de árvores de decisão, $f_k(x_i)$ é a função de entrada na árvore de decisão $k^{th}$, $\hat{y}_i$ é o valor previsto, e $F$ é o conjunto de todas as AD do ensemble. Vários modelos de AD são criados e cada um desses modelos é treinado sequencialmente para que ocorra a minimização do erro. Essa minimização é realizada pelo cálculo do gradiente aplicado a uma função de perda (seja para classificação ou regressão). O XGBoost leva a expansão de Taylor da função de perda até a segunda ordem e adiciona um termo de regularização para encontrar a solução ótima. A função objetivo do XGBoost, dada pela Equação 2, inclui duas partes: erro de treinamento (soma à esquerda) e regularização (soma à direita). Adicionar uma árvore a cada vez é, na verdade, aprender uma nova função $f_k(X,\ theta_k)$ para ajustar o resíduo da última previsão. Quando $K$ árvores são obtidas após o treinamento, os atributos das amostras de predição terão um nó folha correspondente em cada árvore, e cada nó folha corresponde a uma pontuação. Finalmente, as pontuações correspondentes de cada árvore são somadas para obter o valor de predição da amostra (Guo et al. 2020).

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F \qquad (1) \qquad X_{obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \qquad (2)$$

### 2.1. Trabalhos Relacionados

A maioria dos trabalhos encontrados na literatura voltados para o treinamento de algoritmos de AM utilizam arquiteturas com bom poder computacional em GFLOPS, tais como x86 ou GPUs (Serpa et al. 2018). As placas do tipo *single-board computers* baseadas em ARM geralmente são utilizadas para a inferência de modelos de AM (Kanagachidambaresan et al. 2021; Tynes 2021; Kim et al. 2020; Partel et al. 2019; Mittal 2019) e poucos casos na literatura utilizam esse tipo de arquitetura no treinamento de AM (Süzen et al. 2020; Kaewkasi and Srisuruk 2014).

No trabalho de (Süzen et al. 2020) é avaliado o uso de placas *single-board computers* para o treinamento e teste de uma Rede Neural Convolutiva (RNC). Foram utilizadas três *single-boards* (Nvidia Jetson Nano, Nvidia Jetson TX2, and Raspberry Pi 4) para o treinamento utilizado o conjunto de dados *DeepFashion2*. Os resultados mostraram que a Jetson Nano e o Raspberry Pi 4 são capazes de treinar a RNC com conjuntos de dados pequenos (10 mil imagens), mas sem bom desempenho para conjuntos a partir de 20 mil imagens, pois ambas as *single-board* não tiveram memória suficiente. Apenas a Jetson TX2 conseguiu executar todos os conjuntos de dados com tempo de treinamento de 235 segundos e 97,8% de acurácia para o maior conjunto de dados (45 mil imagens).

(Khaydarova et al. 2020) propõem uma arquitetura chamada Rock-CNN para o uso de RNC para aplicações com análise de sentimentos e reconhecimento de imagens

em IoT. Um cluster com 24 RockPro64 *single-board computers* é testada para avaliar o seu desempenho usando o benchmarck Linpack. Os autores sugerem que a arquitetura proposta pode ser usada para aplicações de rede neural, porém, não apresentam nenhum conjunto de treinamento com RNC nesta arquitetura, apenas os testes com Linpack.

O trabalho de (Holt and Sievert 2021) é focado no treinamento do algoritmo Gradiente Descendente Estocástico (GDE) com o PyTorch. Foi utilizada a biblioteca Dask, a qual permite paralelizar um código inteiro, ou parte dele. O trabalho usou uma implementação do GDE específica para GPUs e para tarefa de classificação de imagens com o conjunto de dados CIFAR10. Os resultados mostram que o uso do Dask no treinamento do algoritmo GDE pode reduzir o tempo do treinamento.

No trabalho de (Sapio et al. 2019) é avaliado o desempenho de um processador ARM e um x86 na versão para servidor utilizando a suíte de benchmarck BigDataBench. O objetivo é discutir o potencial do processador ARM para treinamento de AM. Os resultados demonstraram que o desempenho do ARM foi ligeiramente inferior ao x86 para os conjuntos de dados grandes o suficiente para utilizar todos os núcleos da CPU ARM.

(Kaewkasi and Srisuruk 2014) apresenta um estudo de um cluster ARM para processamento de Big Data com 22 *single-board computers* do modelo Cubieboard. Para entender as limitações e restrições do cluster, foram feitos experimentos com três configurações de hardware diferentes com três micro-benchmarks, que foram combinados em um programa para encontrar as palavras mais frequentes. Os resultados experimentais, mostram que o cluster consegue processar um arquivo de 34 GB em tempo aceitável, enquanto o consumo de energia foi de 0,061-0,322 KWh para todos os benchmarks. Foi observado que para função de E/S, o hardware é rápido, mas o poder de processamento dos processadores é inadequado.

Ainda, com exceção do trabalho de (Süzen et al. 2020), não foram encontrados trabalhos que utilizam uma placa *single-board computer* para treinamento e teste de algoritmos de AM. A grande maioria dos trabalhos utilizam as arquiteturas x86 e GPU e as placas *single-board computer* apenas para inferência com o modelo final.

## 3. Objetivos e Metodologia

O principal objetivo deste trabalho é avaliar a viabilidade do uso de *single-board computers* baseadas em ARM para o treinamento de algoritmos de AM, mais especificamente fazendo um estudo utilizando o algoritmo XGBoost. A metodologia adotada consiste em dividir os experimentos em duas fases. Na primeira fase é avaliado o desempenho das *single-board computers* para treinar o algoritmo XGBoost e comparar com outras arquiteturas em tempo de execução, consumo de energia e EDP. Essa comparação foi feita quando ajustando os hiperparâmetros do algoritmo, gerando 36 configurações experimentais, as quais foram executadas em 4 arquiteturas diferentes. Cada configuração foi executada 10 vezes. Na Figura 1 são apresentadas as 36 configurações experimentais, onde para cada configuração é adotada uma combinação de valores de hiperparâmetros. O parâmetro N_jobs define a quantidade de núcleos em paralelo que o processador usará para execução do algoritmo. Quando o valor de N_jobs = -1 todos os núcleos serão utilizados e para N_jobs = 1 apenas um núcleo é usado. O parâmetro ETA é utilizado para controlar o ritmo de aprendizado e define a quantidade de correção feita a cada passo de boosting. Já N_estimators define a quantidade de árvores que serão construídas no pro-

cesso de boosting e Max_depth indica a profundidade máxima de cada árvore. Assim, por exemplo, na configuração 1 o experimento envolve a execução do XGBoost com N_jobs= -1, ETA = 0,3, N_estimators = 100 e Max_depth = 3.

| Configuração | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N_jobs | -1 | | | | | | | | | | | | | | | | | | 1 | | | | | | | | | | | | | | | | | |
| ETA | 0,3 | | | | | | 0,5 | | | | | | 1 | | | | | | 0,3 | | | | | | 0,5 | | | | | | 1 | | | | | |
| N_estimators | 100 | | | 200 | | | 100 | | | 200 | | | 100 | | | 200 | | | 100 | | | 200 | | | 100 | | | 200 | | | 100 | | | 200 | | |
| Max_depth | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 | 3 | 6 | 9 |

**Figura 1. Configurações usadas durante a execução do XGBoost**

Na segunda fase, foram avaliadas apenas as configurações 2, que correspondem à configuração padrão do algoritmo XGBoost (sem ajustar nenhum hiperparâmetro) e a configuração 6 que obteve o melhor resultado (acurácia, tempo e consumo de energia) na primeira fase. Nesta fase foi usado o conjunto de dados com 11 milhões de exemplos e comparados os resultados entre as arquiteturas usando a Equação 3 (Subseção 3.2) para avaliar a viabilidade das arquiteturas *single-board* baseadas em ARM, considerando o tempo de execução, energia, EDP, $CO_2$e e o custo de aquisição da arquitetura.

### 3.1. Arquiteturas, Conjuntos de Dados e Ferramentas

As quatro arquiteturas utilizadas estão detalhadas na Tabela 1. A TX-2 e a Xavier são *single-board computers*. A TX-2 é composta por 4 Jetson TX-2 em cluster e a Xavier por uma Jetson AGX Xavier. Na arquitetura Xavier o algoritmo é executado somente em CPU (Xavier CPU) e somente em GPU (Xavier GPU). CPU Intel é uma arquitetura x86 de alto desempenho com um processador Intel de 8a Geração. A GPU RTX possui as mesmas características da CPU Intel, mas com uma GPU Nvidia GeForce da família Turing.

**Tabela 1. Arquiteturas utilizadas para a execução dos experimentos**

| | TX-2 | Xavier | CPU Intel | GPU RTX |
|---|---|---|---|---|
| **CPU** | NVIDIA Denver 2, Cortex-A57 @ 2GHz | ARM v8.2 64-bit | Intel Core i7 8700 @ 3.2GHz | Intel Core i7 8700 @ 3.2GHz |
| **CPU Cores** | 18C | 8C | 6C 12T | 6C 12T |
| **RAM** | 24GB | 32GB | 64GB | 64GB |
| **GPU** | NVIDIA Pascal @ 1300MHz | NVIDIA Volta | - | Nvidia GeForce RTX 2080Ti @1545 MHz |
| **GPU RAM** | - | - | - | 11GB |
| **GPU Cores** | 768C | 512c | - | 544C |
| **OS** | Ubuntu 18.04 | | Ubuntu 20.04 | |
| **Kernel** | 4.9 | | 5.8 | |
| **Custo** | R$20796 [3] | R$8699 | R$7500 | R$13500 |

Como mencionado, o algoritmo utilizado neste trabalho é o XGBoost (Chen et al. 2015) para a tarefa de classificação implementado em Python.

O conjunto de dados para treinamento do algoritmo foi o *Bóson de Higgs* [4] usado para classificar se um determinado evento é um decaimento de Higgs ou não. O conjunto de dados tem 11 milhões de exemplos e 28 atributos numéricos. Para a primeira fase de experimentos foi gerado um conjunto de dados com 5 milhões de exemplos tirados da base original e mantendo os 28 atributos e para a segunda fase o conjunto completo.

[3] O custo de cada TX-2 foi de R$5199.

[4] https://archive.ics.uci.edu/ml/datasets/HIGGS

As ferramentas utilizadas foram: Dask [5], uma biblioteca *open-source* para computação paralela em Python; nvidia-smi [6], para coletar energia no hardware da NVidia; e o perf [7], para coletar energia na CPU.

A eficiência energética de cada arquitetura foi avaliada pela relação consumo energético e tempo de execução usando a métrica *Energy Delay Product* ($EDP = Energia$ (Joules)$\times Delay$ (segundo)), onde *Delay* é o tempo de execução do algoritmo. O tempo de execução e o consumo de energia são coletados para todas as arquiteturas.

### 3.2. Avaliação do Ganho da Arquitetura

Para avaliar o ganho obtido para cada arquitetura usada nos experimentos foi utilizada a Equação 3 proposta por (Ferro et al. 2017). Essa equação é utilizada para comparar diferentes arquiteturas considerando o tempo de execução e o consumo de energia para um determinado conjunto de aplicações, e o custo de aquisição das arquiteturas avaliadas.

$$G(k) = \sum_{j=1}^{n} w_j[w_d D(j,k) + w_e E_{S(j,k)}] + w_c C_{Ek} \quad (3)$$

Nesta equação $k$ refere-se às $m$ arquiteturas avaliadas, onde $k = (1, 2, ..., m)$; $w_j$ é o peso atribuído a cada aplicação $j$ do conjunto de $n$ aplicações que serão executadas, tal que $\sum_{j=1}^{n} = 1$; $D(j,k)$ é o tempo de execução normalizado da aplicação $j$ na arquitetura $k$; $E_{S(j,k)}$ é o valor normalizado da energia consumida na execução da aplicação $j$ na arquitetura $k$; $C_{Ek}$ é o valor normalizado do custo da arquitetura $k$. Os pesos $w_d$, $w_e$ e $w_c$ são atribuídos ao tempo de execução, consumo de energia e custo da arquitetura respectivamente, caso se queira atribuir diferente relevância a cada parâmetro.

**Tabela 2. Diferentes combinações de pesos - $P(i)$ - atribuídos ao tempo de execução ($W_d$), consumo de energia ($W_e$) e custo da arquitetura ($W_c$)**

| $P(i)$ | $W_d$ | $W_e$ | $W_c$ |
|---|---|---|---|
| **1** | 1/3 | 1/3 | 1/3 |
| **2** | 0,6 | 0,2 | 0,2 |
| **3** | 0,2 | 0,6 | 0,2 |
| **4** | 0,45 | 0,45 | 0,1 |

Na Tabela 2 são apresentados os valores dos pesos $w_d$, $w_e$ e $w_c$ utilizados na avaliação final dos resultados. Foram utilizadas quatro combinações de pesos $P(i)$. Para $i = 1$ os 3 parâmetros são considerados com igual relevância; em $i = 2$ maior relevância ao tempo de execução; $i = 3$ o consumo de energia foi considerado como o parâmetro mais relevante; e $i = 4$ o tempo de execução e o consumo de energia são considerados com igual relevância, porém, o custo da arquitetura é considerado de baixa importância.

### 3.3. Cálculo da emissão de $CO_2$e

Para o cálculo da emissão de $CO_2$e foram utilizadas as Equações 4 e 5 (Strubell et al. 2019; Henderson et al. 2020). Na Equação 4 é calculada a Energia

[5]https://dask.org/

[6]https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pd

[7]https://man7.org/linux/man-pages/man1/perf-stat.1.html

Total (ET) em KWh, onde *PUE (Power Usage Effectiveness*) representa a eficiência energética, t, o tempo em horas, *P_CPU* , *P_GPU* e *P_ MEM*, respectivamente, potência em Watts da CPU, GPU e memória RAM. Essas métricas foram coletadas com a ferramenta perf. Na Equação 5 é calculada a emissão de $CO_2$e, os parâmetros utilizados são: a constante de $CO_2$e do país, a qual de acordo com (Miranda 2012) é de 0,125 Kg/KWh para o Brasil, e a ET.

$$ET = \frac{PUE \times t \times (P\_CPU + P\_GPU + P\_MEM)}{1000} \quad (4) \qquad CO2e = 0,125 \times ET \quad (5)$$

## 4. Experimentos e Resultados

Para os resultados da primeira fase são analisadas a variação no consumo de energia, tempo de execução e EDP quando realizando o ajuste de hiperparâmetros do algoritmo. Esse conjunto de parâmetros foi analisado no trabalho de (Silva et al. 2021), onde se buscou encontrar um ajuste capaz de reduzir o consumo de energia, sem reduzir a precisão do algoritmo. Porém, não foram utilizadas arquiteturas do tipo ARM. Neste artigo o objetivo é verificar como esse mesmo ajuste tem impacto em arquiteturas de baixo consumo de energia e por meio do EDP verificar se este tipo de arquitetura é vantajosa em busca de soluções mais verdes, já que o tipo de arquitetura não influencia na precisão do algoritmo.

Nos gráficos das Figuras 2 e 3 é possível analisar como a variação de hiperparâmetros influenciou no consumo de energia e tempo nas arquiteturas Xavier CPU e CPU Intel[8], para as 36 configurações experimentais. Os valores foram normalizados em relação a configuração padrão do algoritmo (configuração 2). Analisando o impacto na variação do parâmetro ETA (configurações 1-6, 7-12 e 13-18, e se repetindo a cada 6 configurações), este parâmetro não influenciou no tempo de execução e consumo de energia para nenhuma das arquiteturas.

Analisando o parâmetro N-jobs, responsável pela definição do número de núcleos usados em paralela, apesar de parecer óbvio que o aumento do número de núcleos reduz o tempo de execução, essa relação não foi observada para as arquiteturas do tipo GPU. Quando o parâmetro N_jobs = -1 (configurações 1 a 18) houve uma redução de tempo e energia de até 3 vezes para as arquiteturas Xavier CPU e CPU Intel (Figuras 2 e 3). Porém, o mesmo não foi observado para a Xavier GPU, onde houve aumento do tempo de execução e consumo de energia para algumas configurações. E para a GPU RTX até houve uma redução muito pequena quando o parâmetro N_jobs=-1 (usando todos os núcleos) no consumo energético e tempo de execução.

Como observado no trabalho (Silva et al. 2021), a variação dos parâmetros Max_depth e N_estimators são os que tem maior impacto na variação do consumo de energia e tempo. Ao aumentar o valor do parâmetro Max_depth há um aumento no tempo de execução, conforme pode ser observado a cada três configurações dos gráficos. Porém, o aumento do consumo de energia não é proporcional ao aumento do tempo, o que dificulta a análise direta dos resultados obtidos nas diferentes arquiteturas. Assim, foi avaliado o resultado do EDP, para as 36 configurações e comparando toda as arquiteturas (Figura 4). Para a arquitetura TX-2 os resultados não são apresentados a partir da configuração 19, pois não foi possível executar com apenas um núcleo (N_jobs=1) nesta arquitetura.

[8]Os gráficos para as demais arquiteturas não foram apresentadas por limitação de espaço.

O melhor resultado é indicado pelo menor EDP. Os resultados mostram que as arquiteturas Xavier CPU e CPU Intel tiveram o EDP melhor do que Xavier GPU quando o valor de N_jobs = -1. Mas quando N_jobs = 1, a Xavier GPU teve o melhor EDP entre todas arquiteturas. Analisando o gráfico (Figura 4), a Xavier GPU manteve o EDP estável sem um aumento abrupto, como o ocorrido Xavier CPU. Isso se deve ao grande aumento no tempo de execução quando o número de núcleos foi reduzido de 8 para 1, chegando a quase 6x entre as configurações 6 (8 núcleos) e 24 (1 núcleo), enquanto que a variação para a CPU Intel foi de quase 3x.

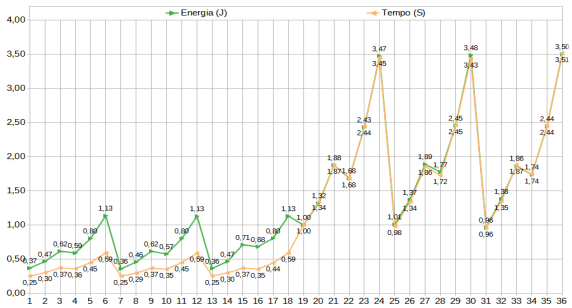


**Figura 2. Energia e Tempo das 36 configurações na arquitetura Xavier CPU.**

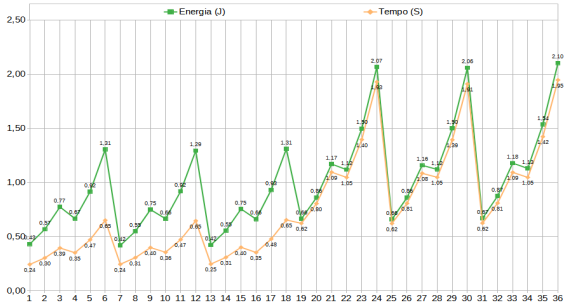


**Figura 3. Energia e Tempo das 36 configurações na arquitetura CPU Intel.**

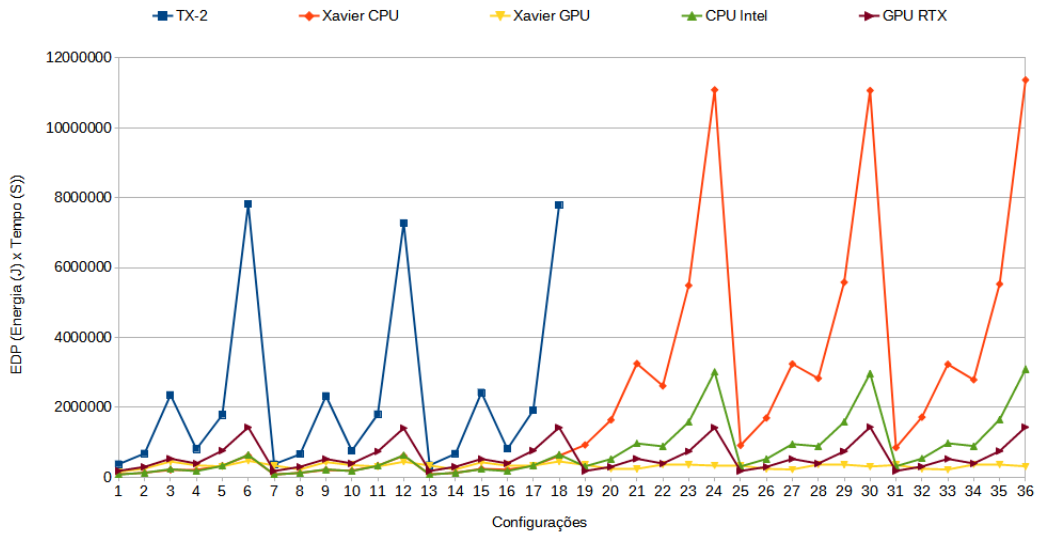


**Figura 4. EDP de todas as arquiteturas para 36 configurações na primeira fase.**

Na comparação entre a primeira e a segunda fase, ou seja, quando utilizando os conjunto de dados de 5 e 11 milhões de exemplos, respectivamente, foi observado um aumento no consumo energético diferente entre as arquiteturas (Tabela 3). As configurações utilizadas na segunda fase tiveram um aumento no consumo energético e no tempo de execução de 1 vezes na CPU Intel e de até 2 vezes na Xavier CPU. A emissão de $CO_2$e na segunda fase foi de até 2 vezes maior do que na primeira fase, a diferença pode ser observada quando comparadas as Figuras 5 e 6. Mas tal emissão é pequena se comparado com o carro popular mais vendido no Brasil. Segundo o Inmetro, o carro emite 0,100

Kg/km de $CO_2$ (motor 1.0 - 8 V, gasolina) enquanto a GPU RTX (arquitetura com pior desempenho na emissão de $CO_2$e) precisa de 4,27 horas de treinamento.

**Tabela 3. Resultados da segunda fase de experimentos**

| Arquitetura | Configuração 2 | | | | Configuração 6 | | | |
|---|---|---|---|---|---|---|---|---|
| | Primeira Fase | | | | | | | |
| | Energia(J) | Tempo(S) | EDP | $CO_2$e | Energia(J) | Tempo(S) | EDP | $CO_2$e |
| TX-2 | 4154,78 | 163,47 | 679181,89 | 0,00024 | 13661,12 | 571,57 | 7808286,36 | 0,00079 |
| Xavier CPU | 1190,40 | 109,87 | 130789,25 | 0,00007 | 2900,03 | 213,94 | 620432,42 | 0,00017 |
| Xavier GPU | 1897,07 | 132,06 | 250527,06 | 0,00011 | 2677,57 | 177,12 | 474251,20 | 0,00016 |
| CPU Intel | 2676,33 | 48,60 | 130077,47 | 0,00016 | 6152,50 | 104,41 | 6545847,68 | 0,00036 |
| GPU RTX | 5521,39 | 54,00 | 298155,06 | 0,00032 | 13051,85 | 109,56 | 1429960,69 | 0,00076 |
| | Segunda Fase | | | | | | | |
| TX-2 | 8144,86 | 309,43 | 2520265,20 | 0,00047 | 21495,62 | 845,60 | 18176696,18 | 0,00125 |
| Xavier CPU | 2508,98 | 234,79 | 589082,35 | 0,00015 | 5895,18 | 439,34 | 2589958,87 | 0,00034 |
| Xavier GPU | 2696,23 | 286,21 | 771673,91 | 0,00016 | 5552,06 | 623,00 | 3458932,97 | 0,00032 |
| CPU Intel | 5647,08 | 108,36 | 611928,88 | 0,00033 | 13095,18 | 223,86 | 2931498,97 | 0,00076 |
| GPU RTX | 8313,63 | 78,20 | 650134,42 | 0,00048 | 20219,83 | 128,68 | 2601807,14 | 0,00117 |

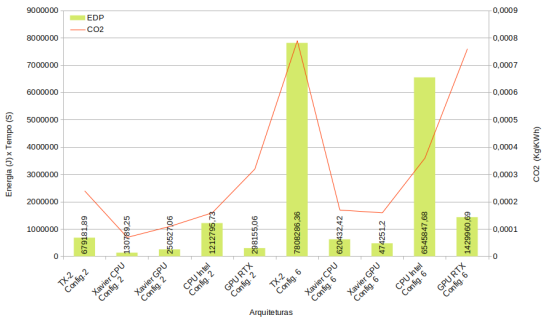


**Figura 5. EDP e $CO_2$ da primeira fase das configurações 2 e 6.**

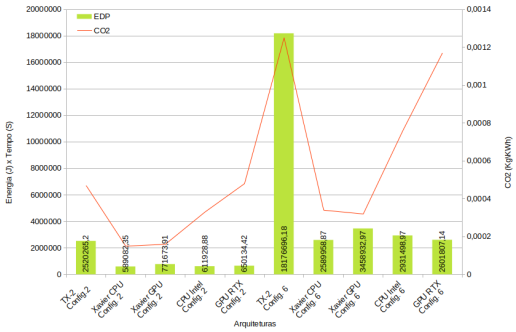


**Figura 6. EDP e $CO_2$ da segunda fase das configurações 2 e 6.**

Comparando o desempenho das diferentes arquiteturas considerando o tempo de execução, consumo de energia e o custo de aquisição, foi utilizada a Equação 3. Cada configuração executada foi considerada como uma aplicação $j$, e todas com a mesma relevância. Assim, os pesos associados a cada configuração são $w_j \simeq \frac{1}{6}, j = \{1, 2, ..., 6\}$.

**Tabela 4. Tempos de execução "$t(j,k)$", consumo de energia "$e(j,k)$", seus respectivos valores normalizados $D(j.k)$ e $E_{S(j.k)}$.**

| $(j,k)$ | $t(j,k)$ | $e(j,k)$ | $D(j,k)$ | $E_{S(j,k)}$ |
|---|---|---|---|---|
| **6,TX-2** | 845,60 | 21495,62 | 0,068349 | 0,089106 |
| **6,Xavier CPU** | 439,34 | 5895,18 | 0,131554 | 0,324909 |
| **6,Xavier GPU** | 623,00 | 5552,06 | 0,092771 | 0,344989 |
| **6,Intel CPU** | 223,86 | 13095,18 | 0,258180 | 0,146267 |
| **6,GPU RTX** | 128,68 | 20219,83 | 0,449146 | 0,094729 |

Na Tabela 4 são apresentados os tempos de execução "$t(j,k)$", em segundos, consumo de energia "$e(j,k)$", em joules, e seus respectivos valores normalizados $D(j.k)$ e $E_{S(j.k)}$, utilizados na função de Ganho, para as configurações $j$ nas arquiteturas $k$, "$(j,k)$", referentes a configuração 6 Fase 2. Pode-se observar que o valor de $E_{S(j,k)}$

para as execuções na Xavier são bem maiores que pra as demais arquiteturas. Este comportamento é similar paras as demais configurações, tanto na fase 1 como na fase 2, isso contribui para que a Xavier tenha uma pontuação mais elevada na avaliação da função ganho (Tabelas 5 e 6), exceto quando o peso atribuído à energia é baixo.

Na Tabela 5 são apresentados os resultados do ganho obtido para todas as arquiteturas, utilizando os resultados da primeira fase, e quando variando a importância (pesos) atribuídos ao melhor tempo de execução, ao consumo de energia ou ao custo, conforme apresentado na Tabela 2. Destacados em vermelho estão o pior valor para cada combinação de pesos, em azul os 2 melhores. Pode-se observar que a TX-2 obteve a pior pontuação em todas as combinações avaliadas. Já no caso da Xavier executando nos núcleos de CPU é possível observar que ela só não esteve entre as 2 melhores pontuações quando foi considerado o tempo de execução com relevância muito superior que o consumo de energia e o custo da arquitetura. Na configuração P(1) a pontuação da Xavier executando em CPU ficou muito próxima da CPU Intel e na configuração P(4) observamos que a GPU RTX teve a melhor pontuação, com a Xavier CPU ficando em segundo com uma pontuação muito próxima. Em P(3) a Xavier tanto com execução em CPU e GPU obteve as duas melhores pontuações com maior destaque para execução em CPU.

**Tabela 5. Ganho de cada arquitetura - G(k) - para execução dos experimentos da fase 1 quando variando a importância dos pesos - P(1) a P(4).**

| **Arq** $k$ | **G(k) P(1)** | **G(k) P(2)** | **G(k) P(3)** | **G(k) P(4)** |
|---|---|---|---|---|
| **TX-2** | 0,091056 | 0,085072 | 0,093642 | 0,088144 |
| **Xavier CPU** | 0,240575 | 0,195389 | 0,287352 | 0,241939 |
| **Xavier GPU** | 0,209093 | 0,172899 | 0,233522 | 0,199009 |
| **CPU Intel** | 0,240703 | 0,256964 | 0,210027 | 0,228347 |
| **GPU RTX** | 0,218573 | 0,289676 | 0,175458 | 0,242562 |

Na Tabela 6 são apresentados os resultados da Equação 3 para todas as arquiteturas avaliadas utilizando os resultados da segunda fase. Em vermelho temos o pior valor para cada combinação de pesos, em azul os 2 melhores valores. Pode-se observar que assim como na primeira fase a TX-2 obteve a pior pontuação em todas as combinações avaliadas. Observando as duas melhores pontuações vemos um comportamento similar ao da primeira fase. No entanto, se observarmos com mais detalhes podemos ver que em P(1) a Xavier CPU obteve uma pontuação um pouco maior do que a CPU Intel. Em P(2) a pontuação das arquiteturas ARM apresentou pouca variação em relação a primeira fase, mas a pontuação da CPU Intel teve uma queda e a da GPU RTX teve um aumento. Em P(3) embora a Xavier GPU tenha se mantido em segundo, a pontuação teve um aumento considerável em relação a primeira fase.

**Tabela 6. Ganho de cada arquitetura - G(k) - para execução dos experimentos da segunda fase quando variando a importância dos pesos - P(1) a P(4).**

| **Arq** $k$ | **G(k) P(1)** | **G(k) P(2)** | **G(k) P(3)** | **G(k) P(4)** |
|---|---|---|---|---|
| **TX-2** | 0,092850 | 0,088928 | 0,094135 | 0,090590 |
| **Xavier CPU** | 0,232414 | 0,191544 | 0,271412 | 0,230810 |
| **Xavier GPU** | 0,221663 | 0,172635 | 0,264259 | 0,216150 |
| **CPU Intel** | 0,230829 | 0,245796 | 0,197257 | 0,214882 |
| **GPU RTX** | 0,222244 | 0,301097 | 0,172937 | 0,247569 |

Assim, com base nos resultados apresentados nas Tabelas 5 e 6, comparando os ganhos de cada arquitetura, é possível observar que a Xavier é uma alternativa viável para o treinamento do XGBoost.

## 5. Conclusão e Trabalhos Futuros

Com os resultados obtidos neste trabalho foi possível avaliar a viabilidade do uso de *single-board computers* do tipo ARM para o treinamento do algoritmo de AM XGBoost. Os resultados mostram que o consumo energético, tempo de execução e a relação entre eles por meio do EDP, para treinamento do XGBoost foram satisfatórios. No caso dos hiperparâmetros, os resultados demonstraram que o N_jobs, Max_depth e N_estimators influenciam no consumo energético e no tempo de execução para todas as arquiteturas. O mesmo foi observado para o aumento na quantidade de exemplos do conjunto de dados. Mesmo para as configurações com maior impacto (maior profundidade da árvore - Max_depth) e com maior número de AD no ensemble foi possível executar os experimentos nas *single-board*, mesmo com 11 milhões de exemplos. Porém, uma limitação deste resultado é que ele não podem ser generalizado para outros algoritmos de AM.

Como trabalhos futuros, será feita uma análise mais aprofundada do desempenho e viabilidade do uso de *single-board computers* para treinamento de outros modelos de AM. Além disso, é necessário a utilização de conjuntos de dados maiores, principalmente para as execuções em GPU. Quanto a co-relação dos hiperparâmetros avaliados com o consumo energético e o tempo de execução, é importante aumentar o intervalo de variação e o número de pontos para os hiperparâmetros Max_depth e N_estimators.

## Agradecimentos

Os autores agradecem ao LNCC/MCTI, CAPES e a FAPERJ pelo apoio financeiro. Este trabalho é parte dos projetos colaborativos CLIMAT-AmSud GreenAI (21-CLIMAT-07), e *Towards a Sustainable Artificial Intelligence - SusAIn* Times Associados INRIA-LNCC.

## Referências

[Chen et al. 2015] Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., et al. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4.

[Ferro et al. 2017] Ferro, M., Silva, G., Klôh, V., Yokoyama, A. M., Mury, A. R., and Schulze, B. (2017). Challenges in HPC evaluation: Towards a methodology for scientific application requirements. In Grandinetti, L., Mirtaheri, S. L., Shahbazian, R., Sterling, T. L., and Voevodin, V. V., editors, *Big Data and HPC: Ecosystem and Convergence, TopHPC 2017, Tehran, Iran, 24-26 April 2017*, volume 33 of *Advances in Parallel Computing*, pages 32–52. IOS Press.

[Guo et al. 2020] Guo, R., Zhao, Z., Wang, T., Liu, G., Zhao, J., and Gao, D. (2020). Degradation state recognition of piston pump based on iceemdan and xgboost. *Applied Sciences*, 10(18).

[Henderson et al. 2020] Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., and Pineau, J. (2020). Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43.

[Holt and Sievert 2021] Holt, J. and Sievert, S. (2021). Training machine learning models faster with dask. *SciPy Conferences*.

[Kaewkasi and Srisuruk 2014] Kaewkasi, C. and Srisuruk, W. (2014). A study of big data processing constraints on a low-power hadoop cluster. In *2014 International Computer Science and Engineering Conference (ICSEC)*, pages 267–272. IEEE.

[Kaggle 2020] Kaggle (2020). State of data science and machine learning 2020. Technical report, . https://www.kaggle.com/kaggle-survey-2020.

[Kanagachidambaresan et al. 2021] Kanagachidambaresan, G. R., Prakash, K. B., and Mahima, V. (2021). *Programming Tensor Flow with Single Board Computers*, pages 145–157. Springer International Publishing, Cham.

[Khaydarova et al. 2020] Khaydarova, R., Fishchenko, V., Mouromtsev, D., Shmatkov, V., and Lapaev, M. (2020). Rock-cnn: a distributed rockpro64-based convolutional neural network cluster for iot. verification and performance analysis. In *2020 26th Conference of Open Innovations Association (FRUCT)*, pages 174–181.

[Kim et al. 2020] Kim, J., Galanopoulos, A., Joseph, J. V., and Kwak, J. (2020). A study on energy-process-latency tradeoff in embedded artificial intelligence. In *2020 Int. Conf. on Inf. and Communication Tec. Convergence (ICTC)*, pages 22–24. IEEE.

[Miranda 2012] Miranda, M. M. d. (2012). *Fator de emissão de gases de efeito estufa da geração de energia elétrica no Brasil: implicações da aplicação da Avaliação do Ciclo de Vida*. PhD thesis, Universidade de São Paulo.

[Mittal 2019] Mittal, S. (2019). A survey on optimized implementation of deep learning models on the nvidia jetson platform. *Journal of Systems Architecture*, 97:428–442.

[Partel et al. 2019] Partel, V., Kakarla, S. C., and Ampatzidis, Y. (2019). Development and evaluation of a low-cost and smart technology for precision weed management utilizing artificial intelligence. *Comp. and electronics in agriculture*, 157:339–350.

[Sapio et al. 2019] Sapio, A., Canini, M., Ho, C.-Y., Nelson, J., Kalnis, P., Kim, C., Krishnamurthy, A., Moshref, M., Ports, D. R., and Richtárik, P. (2019). Scaling distributed ml with in-network aggregation. *arXiv preprint arXiv:1903.06701*.

[Serpa et al. 2018] Serpa, M. S., Krause, A. M., Cruz, E. H., Navaux, P. O. A., Pasin, M., and Felber, P. (2018). Optimizing machine learning algorithms on multi-core and many-core architectures using thread and data mapping. In *2018 26th Euromicro Int. Conf. on Parallel, Dist. and Network-based Processing (PDP)*, pages 329–333. IEEE.

[Silva et al. 2021] Silva, G., Schulze, B., and Ferro, M. (2021). Performance and energy efficiency analysis of machine learning algorithms towards green ai: a case study of decision tree algorithms. Master's thesis, National Lab. for Scientific Computing.

[Strubell et al. 2019] Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650.

[Süzen et al. 2020] Süzen, A. A., Duman, B., and Şen, B. (2020). Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2020 Int. Cong. on Human-Computer Interaction, Optimization and Robotic Applications*, pages 1–5. IEEE.

[Tynes 2021] Tynes, I. L. (2021). Pain recognition performance on a single board computer. Master's thesis, University of South Florida.

[UNESCO 2021] UNESCO, D.-G. (2021). Preliminary report on the first draft of the recommendation on the ethics of artificial intelligence. https://unesdoc.unesco.org/ark:/48223/pf0000374266.locale=en.

[Vinuesa et al. 2020] Vinuesa, R., Azizpour, H., Leite, I., Balaam, M., Dignum, V., Domisch, S., Felländer, A., Langhans, S., Tegmark, M., and Nerini, F. F. (2020). The role of artificial intelligence in achieving the sustainable development goals. *Nature Communications*, 11(233).