Implementações Eficientes de Random Forest em FPGA de Baixo Custo para Internet das Coisas e Computação de Borda

Alysson Silva[1], Olavo Silva[1], Icaro Moreira[1], José A. Nacif[1], Ricardo Ferreira[1]

[1]email: ricardo@ufv.br, Universidade Federal de Viçosa, Brazil

Resumo. *Random Forest é uma abordagem robusta e amplamente utilizada em aprendizado de máquina. Embora existam diversas implementações paralelas em FPGA, não há estudos comparativos entre essas abordagens. Neste trabalho, comparamos implementações baseadas em multiplexadores, equações e tabelas, utilizando diferentes modelos de FPGAs. Demonstramos que, dependendo da ferramenta de síntese utilizada por cada fabricante, um tipo de descrição pode ser mais apropriado. Esta pesquisa propõe uma avaliação sistemática dessas implementações, com foco na redução de recursos para aplicações em computação de borda e Internet das Coisas. Os resultados indicam que é possível obter uma redução de até 43 vezes nos recursos utilizados, sem comprometer a acurácia. Além das técnicas tradicionais, exploramos também quantização, diagramas de decisão binária, algoritmos de agrupamento k-means e Random Forest com dois níveis.*

1. Introdução

A Random Forest é um algoritmo de aprendizado de máquina supervisionado para tarefas de classificação ou regressão. Ela utiliza as previsões combinadas de várias árvores para mitigar o risco de overfitting, introduzindo aleatoriedade na seleção de características e amostragem dos dados. A predição ou inferência na Random Forest pode ser paralelizada [Penha et al. 2023], oferecendo acurácia e desempenho. Ela pode lidar efetivamente com conjuntos de dados com alta dimensionalidade. No entanto, ao visar hardware de baixo custo, é imperativo otimizar o uso de recursos sem comprometer a acurácia.

Existem várias abordagens para Random Forest em FPGA. As principais abordagens são: Multiplexadores, Tabelas e Equações. Apesar das várias abordagens, os trabalhos [Van Essen et al. 2012, Oberg et al. 2012, Amato et al. 2013, Saqib et al. 2013, Qu and Prasanna 2014, Lin et al. 2017, Owaida et al. 2017, Zhao and Chen 2018, Dávila-Rodríguez et al. 2019, Lin et al. 2019, Ikeda et al. 2020, Zhao et al. 2021, Shah et al. 2022, Zhu et al. 2022, Wang et al. 2022, Murtovi et al. 2023] só apresentam uma solução e não comparam duas ou mais abordagens. Por exemplo, é melhor usar multiplexadores ou uma memória ? Para responder a esta questão e outras, este trabalho tem como principais contribuições: **(a)** propor uma comparação das abordagens para identificar qual é mais apropriada para o uso em FPGAs de baixo custo; **(b)** avaliar abordagens realistas, pois muitas utilizam conjuntos de dados pequenos com muitas árvores. O objetivo é mostrar que o hardware é capaz de calcular 100 árvores em paralelo mais rapidamente que uma CPU [Owaida et al. 2017]. Porém, para muitos conjuntos de dados, poucas árvores já apresentam boa acurácia. Neste trabalho, realizamos a seleção da quantidade de árvores na etapa de treinamento buscando uma boa acurácia; **(c)** comparar as técnicas mais recentes que utilizam quantização com k-means [Jinguji et al. 2018]

e remodelagem com diagramas de decisão (BDD); **(d)** avaliar ferramentas e FPGAs de três fabricantes diferentes, mostrando que existem diferenças na forma como o projeto é apresentado, mesmo que a funcionalidade seja equivalente; **(e)** disponibilizar todos os geradores de aceleradores em FPGA em um repositório [UFV 2024] que convertem em Verilog os modelos gerados pela ferramenta scikit-learn [Pedregosa et al. 2011], amplamente usada pela comunidade.

Este trabalho está organizado da seguinte forma: A Seção 2 apresenta as principais implementações: condicionais, equações e tabela. A Seção 3 apresenta as otimizações recentes. A Seção 4 introduz melhorias nas abordagens. As Seções 5 e 6 mostram as avaliações das diversas abordagens em FPGAs de três fabricantes e as principais conclusões.

2. Fundamentos

2.1. Implementações Clássicas

A Figura 1(a) ilustra uma random forest com duas árvores e três classes: A, B e D. O exemplo considera 4 atributos: temperatura (temp), umidade (umi), pressão (pre) e vazão (vaz). Por exemplo, se a temperatura for maior que 20 e a umidade menor que 50, a árvore 1 irá selecionar a classe B. Cada árvore é diferente, e cada nó de decisão possui dois filhos, sendo que a aresta em negrito representa o caminho em que a condição é verdadeira.

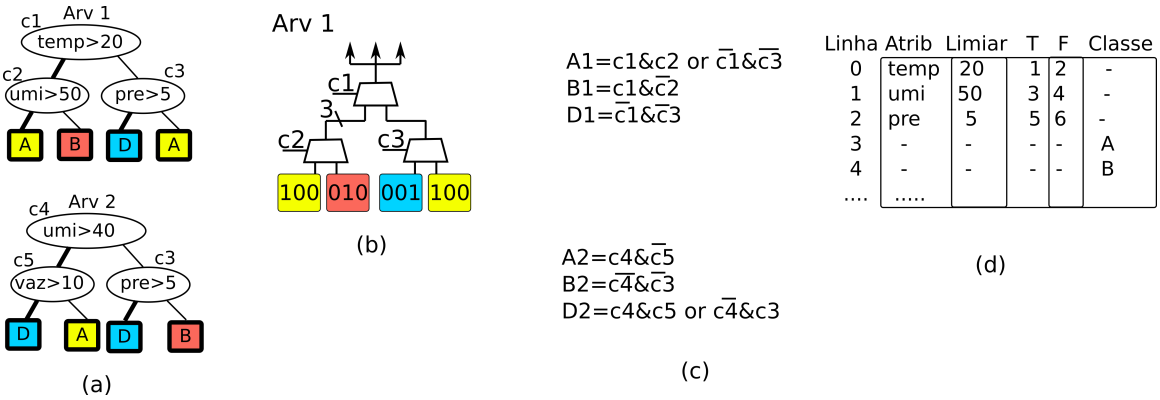

Figura 1. (a) Árvores de Decisão; (b) Multiplexadores; (c) Equações; (d) Tabela.

Existem três principais implementações. A Figura 1(b) mostra a primeira implementação com multiplexadores [Ikeda et al. 2020] para mapear os nós de decisão. Nesse caso, há um mapeamento 1-para-1 da árvore simbólica para a árvore de multiplexadores. Para simplificar a visualização, cada decisão ou comparação é representada por um rótulo c_i. Por exemplo, c_1 representa a comparação $temp > 20$. As classes, em geral, são implementadas com codificação one-hot. Neste exemplo, 100, 010 e 001 representam as classes A, B e D, respectivamente.

A segunda implementação é baseada em equações [Amato et al. 2013], conforme ilustrado na Figura 1(c). Cada caminho até uma folha gera um termo da equação correspondente à classe da folha. Por exemplo, na árvore 1, existem duas folhas com classe A. A saída será da classe A quando a equação $c_1 \& c_2$ ou $\overline{c_1} \& \overline{c_3}$ for verdadeira. Neste exemplo, temos as equações A_1 e A_2 para a classe A nas árvores 1 e 2, respectivamente, ou seja,

cada árvore gera uma equação para cada classe. Essa equação resulta em um voto, e todos os votos são somados para decidir a resposta final, utilizando somadores e comparadores.

A Figura 1(d) ilustra a terceira implementação, baseada em tabelas, que é a mais utilizada em FPGA [Oberg et al. 2012, Van Essen et al. 2012, Zhu et al. 2022]. No exemplo, mostramos uma parte da tabela para à árvore 1, onde nó raiz está na linha 0 da tabela. Teremos um campo para o atributo e outro para o limiar. Em seguida, há os campos True (T) e False (F) para indicar o próximo nó a ser visitado. Neste exemplo, esses campos apontam para as linhas 1 e 2, que correspondem aos dois nós internos. A tabela também armazena os nós folha; neste exemplo, apenas as duas primeiras folhas estão ilustradas nas linhas 3 e 4 da tabela. A vantagem do uso de tabelas é a facilidade de reconfiguração, caso sejam gravadas em uma memória. Dessa forma, não é necessário repetir o passo de síntese ao alterar o modelo, um processo normalmente demorado no projeto de FPGAs.

A primeira contribuição deste trabalho é um gerador de código Verilog a partir de um modelo criado pela biblioteca Scikit-learn [Pedregosa et al. 2011]. Diferentemente de trabalhos anteriores, que focam em apenas uma implementação, nossa abordagem permite comparar os recursos necessários para cada uma das três principais abordagens. Observamos que, para as ferramentas de FPGA avaliadas, o resultado da otimização depende do código Verilog de entrada. Portanto, é essencial desenvolver ferramentas de geração automática para explorar diferentes possibilidades, uma vez que, embora os FPGAs sejam semelhantes, as ferramentas de síntese tomam decisões distintas ao otimizá-los.

2.2. Votação

O próximo passo é a votação. Utilizando a codificação one-hot, podemos somar o bit correspondente a cada classe de cada árvore, conforme ilustrado na Figura 2(a). Supondo o exemplo com as classes A, B e D, e n árvores, serão necessários três somadores de $\log n$ bits para calcular a votação de cada classe. Em seguida, um circuito determinará a classe vencedora e não requer codificação one-hot. Podemos, por exemplo, definir os códigos 00, 01 e 10 para representar as classes A, B e D, respectivamente.

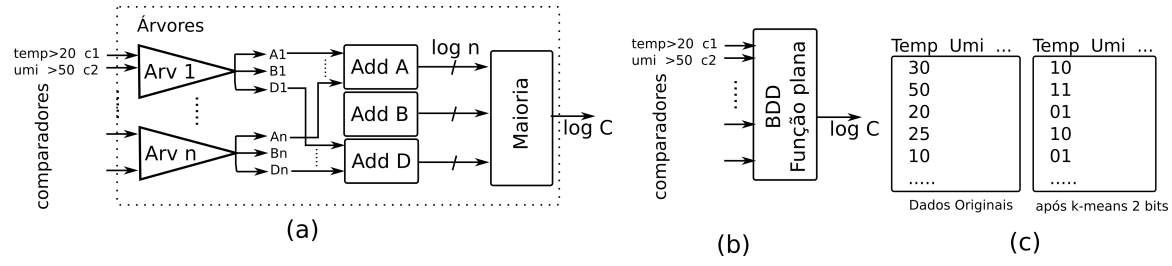

Figura 2. (a) Diagrama completo com Árvores e Votação; (b) Função Plana com BDD; (c) Quantização com K-means.

3. Otimizações

A síntese em FPGA pode simplificar o circuito de forma a depender apenas das variáveis primárias. Outra possibilidade é realizar essa otimização utilizando diagramas de decisão binária (BDDs)[Nakahara et al. 2017, Murtovi et al. 2023, Silva et al. 2023] e, em seguida, repassar o BDD para a ferramenta de FPGA. Uma random forest pode ser vista como uma função booleana que depende das entradas $c_1, \ldots, c_m$, onde m é o número de

comparadores gerados pelos nós das árvores. O BDD faz uma representação plana (ou flat), como ilustrado na Figura2(b), mesclando as equações das árvores com a lógica do somador e da função de maioria. Isso gera uma função global que pode ser otimizada de forma mais eficiente. Outra vantagem do BDD é a ordenação das variáveis. Em uma random forest, uma comparação de variável c_i pode aparecer em qualquer nível e em qualquer árvore. Já no BDD, existe uma ordem fixa, o que facilita o roteamento dos sinais de comparação no circuito final.

Uma desvantagem dos BDDs [Silva et al. 2023] é o tempo de conversão e o tamanho do diagrama final quando há muitos comparadores. Uma forma de mitigar esse problema é utilizar a quantização com k-means para as árvores, conforme proposto em [Jinguji et al. 2018]. A ideia é mapear cada atributo de entrada para uma codificação quantizada com k-means, como ilustrado na Figura 2(c), onde o atributo Temperatura é agrupado em quatro códigos com k-means: 00, 01, 10 e 11, reduzindo assim o número de comparações necessárias. Neste trabalho, aplicamos a técnica de quantização em conjunto com os BDDs, mas outras técnicas também podem ser exploradas [Bueno et al. 2024b].

O gerador proposto neste trabalho também incorpora a criação do BDD a partir do scikit-learn [Pedregosa et al. 2011], além da possibilidade de inclusão do k-means para codificação. O objetivo é comparar as três implementações clássicas e suas versões otimizadas.

4. Random Forest Multinível

Avaliamos também, novas implementações que são combinações das técnicas anteriores. A Figura 3(a) mostra o uso de k-means para atuar em um conjunto de atributos para redução mais agressiva. No exemplo, dois conjuntos são formados: TU e PV, para temperatura e umidade (TU) e pressão e vazão (PV). Para cada conjunto, executamos o k-means com $k = 4$, que irá gerar 4 grupos codificados em 2 bits. Assim, uma nova tabela de dados é construída e sua respectiva random forest, que será bem mais simples com 4 bits de entrada em comparação com a tabela original que tem 4 atributos de 16 bits, ou 64 bits de entrada. A vantagem desta abordagem é que, se a perda de acurácia for pequena, temos uma boa redução do custo das árvores, principalmente para a técnica de BDD.

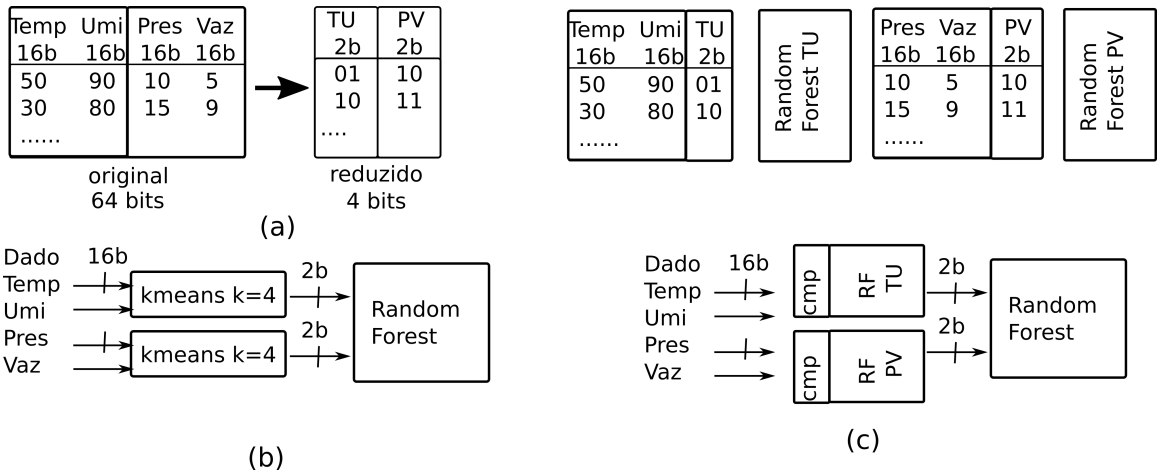

Figura 3. (a) Redução Conjunto de Dados de 64 bits para 4 bits com K-means; (b) Dois níveis: K-means, Random Forest; (c) Dois níveis de Random Forest.

Apesar da quantização reduzir a random forest, na inferência temos que considerar o custo do k-means do circuito que irá implementar o k-means. Pois no dispositivo de IoT, os dados brutos serão aplicados, codificados com o k-means e classificados com a random forest. Portanto, o custo-benefício da redução deve considerar os recursos para o k-means e para a random forest BDD, por exemplo, como ilustrado na Figura 3(b). Para cada atributo temos k subtratores, k multiplicadores e k comparadores. Para o exemplo anterior teremos 16 subtratores, 16 multiplicadores e 16 comparadores. Estes dados são importantes, pois o custo do circuito de pré-processamento pode ser elevado.

Uma alternativa é substituir o k-means por uma random forest, como ilustrado na Figura 3(c). Teremos assim dois níveis de random forest. O primeiro nível faz um pré-processamento e o segundo nível finaliza a classificação. Primeiro, o k-means é executado para criar a tabela intermediária. Depois, para cada tabela intermediária, montamos uma random forest. Os resultados das random forests de primeiro nível irão alimentar a random forest final de segundo nível.

5. Resultados Experimentais

5.1. Ferramentas de Síntese, FPGAs e Conjunto de Dados

Tabela 1. FPGAs: Artix-7, Cyclone IV e GW2AR-18.

Fabricante	FPGA	Luts	Regs	DSP	BRAM
Xilinx	Artix 7-35T	20.800	41.600	90	50
Intel	Ciclone IV EP4CE115	114.480	114.480	266	66
Gowin	GW2AR-18	20.736	15.750	48	46

A primeira contribuição deste trabalho é observar o comportamento de diversas ferramentas de síntese para FPGA em função da descrição Verilog. Todos os códigos gerados foram sintetizados com as ferramentas Vivado da Xilinx, Quartus da Altera/Intel e Gowin Design 1.9.9 Education. Este é o primeiro trabalho que avalia a ferramenta Gowin. Selecionamos FPGAs de baixo custo pois nosso alvo são aplicações IoT: Artix-7 da Xilinx, Cyclone IV da Intel, e GW2AR-18 da Sipeed. A Tabela 1 apresenta os recursos dos FPGAs avaliados.

Tabela 2. Conjunto de dados da base UCI.

Conjunto de Dados	Atributos		Amostras	Classes
	Numérico	Categórico		
Adult	6	8	48.842	2
Susy	18	0	5 milhões	2
Covtype	10	44	581 mil	7
Drybean	16	0	13.611	7

Diferente de muitos trabalhos para FPGA que foram validados com conjuntos de dados pequenos com 1.000 amostras ou menos [Saqib et al. 2013, Nakahara et al. 2017, Jinguji et al. 2018], selecionamos um conjunto de dados representativo do repositório UCI [Asuncion and Newman 2007], que é a referência principal na área de aprendizado de máquina, variando de 10 mil até 5 milhões de amostras. A Tabela 2 apresenta os quatro conjuntos de dados utilizados.

5.2. Resultados das Sínteses

A Figura 4 ilustra o resultado em consumo de Luts nos FPGAs para os quatro conjuntos de dados e as três ferramentas. Optamos por mostrar todos os dados em conjunto para visualizar algumas propriedades. Como temos muitas informações na mesma Figura, iremos avaliar cada ferramenta em separado. A descrição com multiplexadores usou duas abordagens. A primeira com o comando **(cond)?T:F;** em Verilog que é a implementação explícita de multiplexadores. A segunda abordagem utilizou os comandos aninhados de **IF ELSE**. Por exemplo, a descrição da árvore 1 na Figura 1:

```
assign C = (c1)? (c2)?A:B : (c3)?D:A;
// or
if (c1)
   if (c2)   C=A;
   else      C=B;
else
   if (c3)   C=D;
   else      C=A;
```

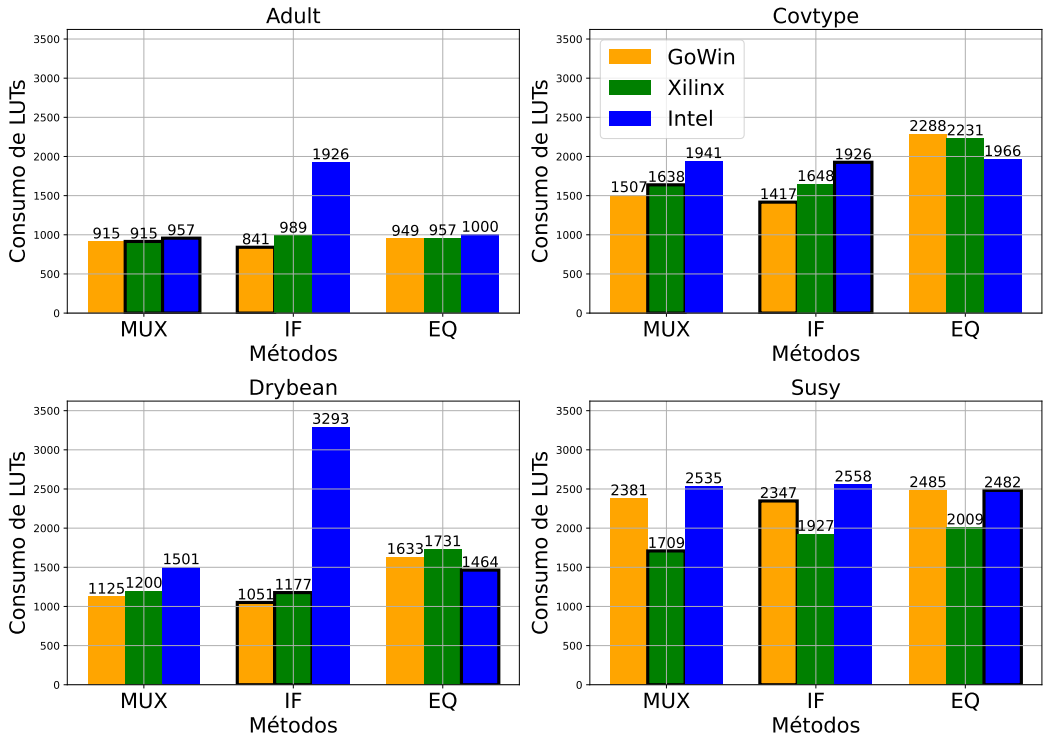

Figura 4. Avaliação de 4 conjuntos de dados nas ferramentas comerciais: Vivado/Xilinx, Quartus/Intel, Gowin/Sipeed. O código é baseado em Multiplexadores (Mux), If-Else (IF), Equações (EQ).

Começando pela ferramenta GoWin (em amarelo), podemos notar que a descrição utilizando *if-else* é a mais apropriada. Usamos uma borda em negrito para destacar a descrição com menor custo para cada ferramenta. Comparando a abordagem *if-else* com multiplexadores, observamos valores próximos, em média 3% maiores. Já a descrição por equações teve desempenho inferior para os conjuntos de dados *covtype* e *drybean*.

Na ferramenta da Intel (em azul), a descrição por equações foi a melhor para os conjuntos *drybean* e *susy*, e ficou bem próxima das melhores para o conjunto *adult*

(onde os multiplexadores foram superiores) e para *covtype* (onde *if-else* foi superior). O desempenho do *if-else* foi ruim para os conjuntos *drybean* e *adult*. Ou seja, enquanto a Intel lida bem com descrições por equações, pode apresentar dificuldades com *if-else*. Por outro lado, a GoWin tem um comportamento oposto. O gerador desenvolvido permite que dependendo do FPGA utilizado, o usuário pode testar rapidamente qual é a melhor opção.

Por fim, analisando os resultados para a ferramenta Xilinx (em verde), o multiplexador apresentou os melhores resultados. Isso demonstra que cada ferramenta tem preferência por um estilo de descrição diferente, apesar de todas serem funcionalmente equivalentes. Como os FPGAs organizam suas Luts de maneiras distintas, não é possível determinar uma única ferramenta como a melhor para a redução de Luts. No entanto, ao agrupar todos os resultados, podemos observar que, apesar das diferenças nas arquiteturas, alguns conjuntos de dados produziram resultados semelhantes para uma técnica específica, independentemente do FPGA.

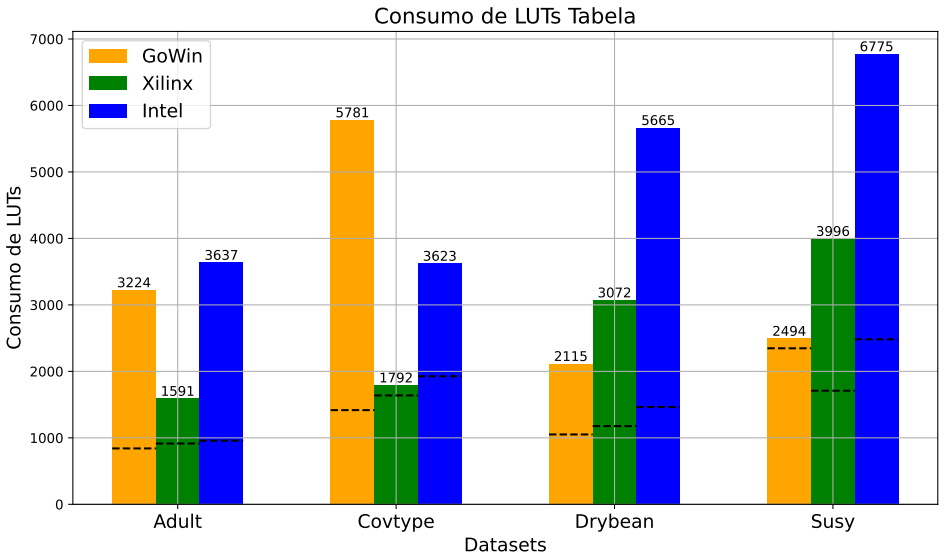

Figura 5. Avaliação de 4 conjuntos de dados nas ferramentas comerciais: Vivado/Xilinx, Quartus/Intel, Gowin/Sipeed. O código é baseado Tabela.

A Figura 5 representa o consumo de Luts para os quatro conjuntos de dados nas três ferramentas, com o código baseado em tabela. Apesar das tabelas serem a abordagem mais usada na literatura, não é a mais apropriada sem personalização do código para as ferramentas. As linhas pontilhadas referenciam os menores consumos utilizando os outros métodos representados na Figura 4. Apenas a ferramenta da Xilinx para Covtype e GoWin para Susy apresentam um resultado próximo das outras descrições. As tabelas são mais apropriada para implementações de alto desempenho, mas deve ser personalizada usando diretivas das ferramentas. No cenário de alto desempenho a reconfiguração dinâmica é fundamental, o que não é o caso para muitos cenários de IoT.

Neste trabalho também avaliamos a ferramenta Yosys de código livre para síntese de FPGA. Não avaliamos o FPGA da Gowin pois a ferramenta ainda está em desenvolvimento. A Figura 6 além das três abordagens: Mux, if-else e equações, apresentam a opção da otimização *flat*. O Yosys possui várias opções de otimização. Exploramos a opção *flat* para comparar com as versões com BDD, que busca ter uma visão planar da

lógica do circuito. Na parte de cima avaliamos o FPGA da Intel. Observamos que os resultados em vermelho, sem a planarização das funções Booleanas, que todas as opções são equivalentes, exceto o if-else para o Drybean. Ao usarmos a opção flat, o circuito tem o custo reduzido, em média, pela metade. Para Intel, o if-else também tem um maior custo com flat para o Drybean. Para o FPGA da Xilinx, tivemos resultados semelhantes, pois depende mais da ferramenta de síntese que do FPGA alvo. A opção flat gerou uma boa redução e o if-else não foi bem capturado para o conjunto de dados Drybean.

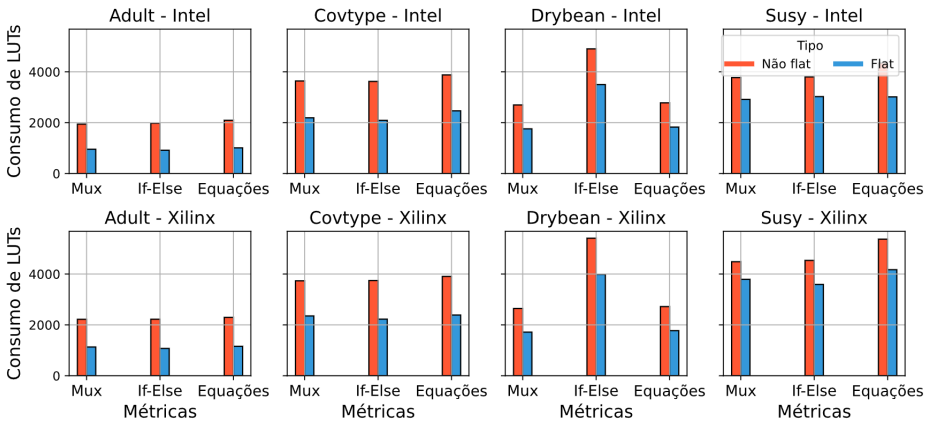

Figura 6. 4 conjuntos de dados utilizando a ferramenta de código aberto Yosys tendo as FGPAs da Xilinx e da Intel como alvo.

Uma contribuição deste trabalho é fornecer um gerador de código compatível com as ferramentas de aprendizado de máquina como scikit-learn [Pedregosa et al. 2011], permitindo ao projetista explorar rapidamente qual descrição é mais efetiva para redução do custo da implementação em FPGA. Por fim, todas as abordagens podem ser implementadas em pipeline, gerando uma avaliação por ciclo, quando for necessário otimizar o desempenho além dos recursos utilizados nos FPGAs.

5.3. Quartis para Quantização e BDDs para Random Forest

Uma abordagem mais simples de quantização é o uso de quartis. Como veremos adiante, os comparadores têm impacto no custo, que pode ser reduzido com os quartis. Para avaliar o uso dos quartis, fizemos um experimento com o conjunto de dados *adult*, que possui 6 atributos numéricos e 8 categóricos. Mesmo com a quantização, a acurácia original foi preservada em 82%. Os quartis transformam cada atributo numérico de 32 bits em 2 bits. Portanto, reduzimos a entrada para o treinamento dos atributos numéricos da random forest de $6 \cdot 32 = 192$ bits para $6 \cdot 2 = 12$ bits, e os atributos restantes são categóricos, sendo 1 de 6 bits, 1 de 5 bits, 3 de 4 bits, 2 de 3 bits e um binário. Assim, reduzimos de 222 bits para 28 bits.

Outro ponto importante é explorar a construção da random forest com as ferramentas de aprendizado de máquina. Para evitar o uso excessivo de árvores, comum nos trabalhos de FPGAs para gerar desempenho em relação à CPU, fizemos uma exploração para encontrar o mínimo necessário de árvores sem perda de acurácia. A configuração com 7 árvores e profundidade 7 manteve a acurácia em 82,3%. Com apenas 28 bits, ou 28 variáveis binárias, a técnica de BDD é atrativa. Apesar de gerar um BDD com 1.317

nós de decisão, a síntese em FPGA reduziu para um circuito com apenas 276 Luts de 6 entradas usando a ferramenta da Xilinx. No entanto, temos que considerar o custo dos comparadores para gerar os quartis. Mesmo incluindo os comparadores, o custo total da síntese ainda é otimizado, com 309 Luts, o que é 3 vezes menor que os melhores resultados sem a quantização/BDD, onde a Random Forest busca otimizar o conjunto de dados original, como ilustrado na 3.

Tabela 3. Otimização com Quartis e BDD para Conjunto de Dados Adult.

BDD com Quartis		Random Forest Original	
Só BDD	BDD e Comparadores	Multiplexadores	Descrição IF/ELSE
100 Luts	309 Luts	915 Luts	989 Luts

5.4. Redução de Dimensionalidade

A redução de dimensionalidade é um pré-processamento utilizado em aprendizado de máquina. Selecionamos o dataset Susy, que possui 5 milhões de amostras e 18 atributos, para avaliar o impacto na acurácia e no custo do hardware após o uso da redução de dimensionalidade. É importante lembrar que qualquer etapa de pré-processamento deve ser incluída no hardware, que sempre lerá os dados brutos em ambientes de IoT.

Em uma primeira etapa, avaliamos a redução do número de atributos de 18 para 6. Para a seleção dos 6 atributos, geramos 100 grupos aleatórios e selecionamos aquele com melhor acurácia. Depois, dividimos o grupo em três conjuntos de 2 atributos. Para cada conjunto, aplicamos um k-means com $k = 5$, ou seja, cada conjunto foi recodificado com 3 bits para 2 atributos, totalizando 9 bits para os 6 atributos.

Inicialmente, fizemos uma avaliação do custo do k-means como primeira etapa, conforme ilustrado na Figura 3(b). Entretanto, o custo dos 90 operadores aritméticos com 3 conjuntos de 2 atributos e $k = 5$ foi próximo ao custo da implementação em um nível. Para reduzir o custo, optamos por treinar outra random forest para substituir o k-means, utilizando dois níveis de random forest, como mostrado na Figura 3(c).

O circuito final terá três random forests no primeiro nível, uma para cada conjunto de 2 atributos, gerando 3 bits de saída cada (ou 5 classes, pois usamos $k = 5$), totalizando 9 bits. No segundo nível, a random forest de classificação será mais simples, recebendo apenas 9 variáveis. A Figura 7(a) mostra o custo do primeiro e do segundo nível, considerando a implementação das random forests com BDDs, que variam de 202 a 262 Luts para cada uma das três random forests, sendo o custo dominado pelos comparadores. O segundo nível possui apenas o BDD com 6 Luts para a random forest de classificação. O custo total é menor que a soma, pois a síntese em FPGA otimiza o circuito globalmente, reduzindo-o para 578 Luts. Em comparação com a versão da random forest original com Mux, que teve um custo de 1.709 Luts, houve uma redução de aproximadamente 3 vezes.

Entretanto, o custo dos comparadores de cada random forest de primeiro nível acaba por mascarar o ganho dos BDDs. Para as mesmas 6 variáveis, se optarmos pela quantização com quartis em 2 bits, totalizando 12 bits ao todo, conforme ilustrado na Figura 7(b), temos uma árvore final com BDD com um custo de apenas 203 Luts, que é mais de 8 vezes menor que a melhor implementação da random forest original. A random forest original também foi restrita à 7 árvores para gerar circuitos compactos.

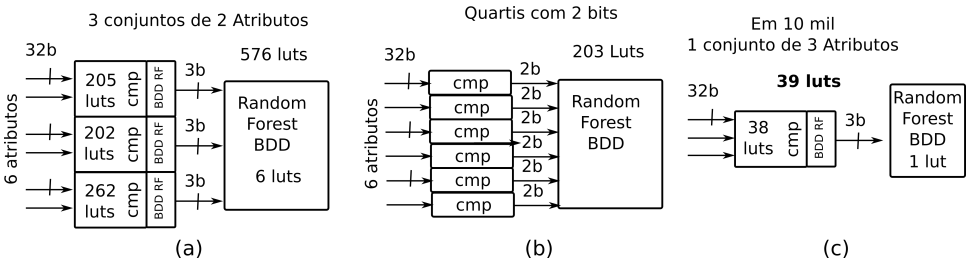

Figura 7. Conjunto de dados Susy:(a) 2 níveis de Random Forest; (b) Quartis + Random Forest; (c) Melhor de 10 mil com 3 Atributos com 3 bits em dois níveis.

Finalmente, fizemos uma terceira opção para o conjunto de dados Suzy. Geramos 10 mil conjuntos de 3 atributos. Para cada conjunto, executamos o k-means com $k = 8$, ou seja, 3 bits para codificar. Construímos as random forests e selecionamos o melhor "k-means". A acurácia foi de 76,4% contra a acurácia de 77,9% do conjunto original. Construímos o BDD de primeiro nível. Com apenas 3 variáveis, o BDD de segundo nível ocupa apenas 1 Lut. O circuito final requer apenas 39 Luts. Portanto é 43 vezes menor que a melhor implementação com Mux que considerou o conjunto completo de 18 atributos.

5.5. BDDs e Implementações por Memória

Os FPGAs de baixo custo têm pouca capacidade de memória dedicada em hardware como as Block-Ram ou BRAMs. Entretanto, para FPGAs de alto desempenho, a implementação por memória pode ser interessante. Outra contribuição deste trabalho foi avaliar o impacto de redução de dimensionalidade e BDDs nos modelos de memória. Dos conjuntos de dados avaliados, selecionamos o *covtype* com 54 atributos, sendo 10 numéricos e 44 categóricos. Em função da grande quantidade de dados categóricos usamos o método k-medoids no lugar do k-means para agrupamentos e redução de dimensionalidade. A métrica de distância usada para os dados categóricos foi distância de hamming, onde foi realizada uma redução de 54 atributos para 20 bits.

Usamos 7 árvores com profundidade ilimitada no treinamento com uma acurácia de 74%. A random forest resultante requer 20.467 nós. Ao aplicarmos a construção do BDD [Silva et al. 2023], obtivemos uma redução para 2.352 nós de decisão no BDD. O BDD tem no máximo 20 níveis e pode ser utilizado em pipeline. Seu uso pode ser indicado para a implementação em tabela em ambiente com FPGA de alto desempenho com uma redução de 10 vezes no tamanho da tabela e uma ordenação no acesso as variáveis que simplifica o roteamento interno dos dados.

6. Conclusão

Embora uma random forest possa ser vista como uma função Booleana, o custo no FPGA depende de vários fatores, como a descrição do projeto, a ferramenta de síntese e o FPGA. Este estudo avalia várias abordagens: multiplexadores, equações, tabelas, BDDs, quantização e random forest com dois níveis. Além disso, analisamos o desempenho das ferramentas de FPGA para mapeamento em diferentes dispositivos. Para as abordagens tradicionais, cada ferramenta preferiu um estilo diferente. Porém para reduções mais agressivas é necessário usar quantização e/ou BDDs, o que é fundamental quando direcionadas para FPGAs de baixo custo. Adicionalmente, neste trabalho, demonstramos uma abordagem em duas etapas, envolvendo redução de dimensionalidade ou quantização, que

pode alcançar uma redução significativa no tamanho, mantendo a acurácia. No entanto, o tamanho do random forest pode ser influenciado por diversos fatores. Pesquisas futuras se concentrarão em explorar os custo-benefícios entre a etapa de redução, implementações de K-means em FPGA [Penha et al. 2018, Bragança et al. 2021, Bueno et al. 2024a] e a etapa de classificação. Dispositivos IoT exigem inferência rápida e adaptabilidade, destacando a importância de uma abordagem multinível em FPGA de baixo custo para alcançar reduções significativas.

Agradecimentos

Apoio financeiro da FAPEMIG APQ-01577-22, CNPq e UFV. Este trabalho também foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

Amato, F., Barbareschi, M., Casola, V., Mazzeo, A., and Romano, S. (2013). Towards automatic generation of hardware classifiers. In *Algorithms and Architectures for Parallel Processing (ICA3PP)*. Springer.

Asuncion, A. and Newman, D. (2007). Uci machine learning repository.

Bragança, L., Canesche, M., Penha, J., Comarela, G., Nacif, J. A. M., and Ferreira, R. (2021). An open source custom k-means generator for aws cloud fpga accelerators. In *Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE.

Bueno, W., , Barros, O., Nacif, J., and Ferreira, R. (2024a). Implementação paralela de múltiplos k-means em gpu. In *Simpósio em Sistemas Computacionais de Alto Desempenho*.

Bueno, W., da Silva, O., Nacif, J., and Ferreira, R. (2024b). Redução de dimensionalidade para Árvores aleatórias. In *Workshop de Iniciação Científica - Simpósio em Sistemas Computacionais de Alto Desempenho*.

Dávila-Rodríguez, I.-A., Nuño-Maganda, M.-A., Hernández-Mier, Y., and Polanco-Martagón, S. (2019). Decision-tree based pixel classification for real-time citrus segmentation on fpga. In *Int Conf on ReConFigurable Computing and FPGAs*. IEEE.

Ikeda, T., Sakurada, K., Nakamura, A., Motomura, M., and Takamaeda-Yamazaki, S. (2020). Hardware/algorithm co-optimization for fully-parallelized compact decision tree ensembles on fpgas. In *Applied Reconfigurable Computing(ARC)*. Springer.

Jinguji, A., Sato, S., and Nakahara, H. (2018). An fpga realization of a random forest with k-means clustering using a high-level synthesis design. *IEICE TRANSACTIONS on Information and Systems*, 101(2):354–362.

Lin, X., Blanton, R. S., and Thomas, D. E. (2017). Random forest architectures on fpga for multiple applications. In *Great Lakes Symposium on VLSI*.

Lin, Z., Sinha, S., and Zhang, W. (2019). Towards efficient and scalable acceleration of online decision tree learning on fpga. In *IEEE FCCM*.

Murtovi, A., Bainczyk, A., Nolte, G., Schlüter, M., and Steffen, B. (2023). Forest gump: a tool for verification and explanation. *International Journal on Software Tools for Technology Transfer*, pages 1–13.

Nakahara, H., Jinguji, A., Sato, S., and Sasao, T. (2017). A random forest using a multi-valued decision diagram on an fpga. In *IEEE Symposium on multiple-valued logic*.

Oberg, J., Eguro, K., Bittner, R., and Forin, A. (2012). Random decision tree body part recognition using fpgas. In *Field Programmable Logic and Applications (FPL)*. IEEE.

Owaida, M., Zhang, H., Zhang, C., and Alonso, G. (2017). Scalable inference of decision tree ensembles: Flexible design for cpu-fpga platforms. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Penha, J., da Silva, A., Barros, O., Moreira, I., Nacif, J., and Ferreira, R. (2023). Avaliação de estilos de código para árvores de decisão em gpu com microbenchmarks. In *Simpósio em Sistemas Computacionais de Alto Desempenho*.

Penha, J. C., Bragança, L., Canesche, M., Comarela, G., Nacif, J. A. M., and Ferreira, R. (2018). A gpu/fpga-based k-means clustering using a parameterized code generator. In *Symp on High Performance Computing Systems (WSCAD)*. IEEE.

Qu, Y. R. and Prasanna, V. K. (2014). Scalable and dynamically updatable lookup engine for decision-trees on fpga. In *High Performance Extreme Computing Conf.* IEEE.

Saqib, F., Dutta, A., Plusquellic, J., Ortiz, P., and Pattichis, M. S. (2013). Pipelined decision tree classification accelerator implementation in fpga (dt-caif). *IEEE Transactions on Computers*, 64(1):280–285.

Shah, M., Neff, R., Wu, H., Minutoli, M., Tumeo, A., and Becchi, M. (2022). Accelerating random forest classification on gpu and fpga. In *Proceedings of the 51st International Conference on Parallel Processing*, pages 1–11.

Silva, O. A., Silva, A. K., Moreira, Í. G., Nacif, J. A., and Ferreira, R. S. (2023). Rdsf: Everything at same place all at once-a random decision single forest. In *2023 XIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–6. IEEE.

UFV (2024). Geradores de código para random forest. https://github.com/arduinoufv/randomforest.

Van Essen, B., Macaraeg, C., Gokhale, M., and Prenger, R. (2012). Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga? In *IEEE FCCM*.

Wang, H., Wu, Z., Wang, X., Bian, L., and Jin, H. (2022). Hardgbm: A framework for accurate and hardware-efficient gradient boosting machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Zhao, S. and Chen, S. (2018). A discretization method for floating-point number in fpga-based decision tree accelerator. In *IEEE Conf on Computer and Communications*.

Zhao, S., Chen, S., Yang, H., Wang, F., and Wei, Z. (2021). Rf-risa: A novel flexible random forest accelerator based on fpga. *Journal of Parallel and Distributed Computing*.

Zhu, M., Luo, J., Mao, W., and Wang, Z. (2022). An efficient fpga-based accelerator for deep forest. In *Int Symp on Circuits and Systems (ISCAS)*. IEEE.