

Arquitetura de FPGA para Inferência Bit-Exata de GBDTs com Árvores Simétricas

Marcus V. S. Nunes^{1,2}, Marcelo A. C. Fernandes^{1,2,3}

¹InovAI Lab, nPITI/IMD, UFRN, 59.078-900, Natal, RN, Brasil

²Leading Advanced Technologies Center of Excellence (LANCE)
nPITI/IMD, UFRN, 59.078-900, Natal, RN, Brasil

³Departamento de Engenharia da Computação e Automação (DCA), UFRN
59.078-900, Natal, RN, Brasil

marcus.vinicius.nunes.122@ufrn.edu.br, mfernandes@dca.ufrn.br

Abstract. *This work proposes a method for mapping Gradient Boosted Decision Tree (GBDT) ensembles of symmetric trees directly to FPGA lookup table (LUT) primitives. The architecture combines a shared comparator bank, leaf-value ROMs, and a balanced adder tree, forming a single combinational inference path with optional pipelining. The proposed method integrates hardware-aware predicate regularization, multi-output trees with shared splits, quantization-aware training, adaptive per-round precision, and post-training ROM optimizations to reduce hardware cost while preserving predictive performance. An automated flow generates synthesizable VHDL and validates bit-exact equivalence through GHDL simulation. On JSC, the architecture achieves 74.56% accuracy using 799 LUTs, while on MNIST it reaches 96.44% accuracy with 4,495 LUTs, demonstrating competitive accuracy with low FPGA resource utilization.*

Resumo. *Este trabalho propõe um método para mapear diretamente ensembles de Gradient Boosted Decision Trees (GBDT) com árvores simétricas em primitivas LUT (lookup table) de FPGA. A arquitetura combina um banco de comparadores compartilhado, ROMs de valores de folha e uma árvore de somadores balanceada, formando um caminho combinacional de inferência com pipeline opcional. O método proposto integra regularização de predicados orientada ao hardware, árvores multi-output com splits compartilhados, quantização integrada ao treinamento, precisão adaptativa por iteração e otimizações pós-treinamento das ROMs para reduzir o custo em hardware, preservando o desempenho preditivo. Um fluxo automatizado gera VHDL sintetizável e valida equivalência bit a bit por simulação GHDL. No JSC, a arquitetura atinge 74,56% de acurácia com 799 LUTs e, no MNIST, 96,44% com 4.495 LUTs, demonstrando acurácia competitiva com baixo uso de recursos de FPGA.*

1. Introdução

O gatilho de Nível-1 do *Large Hadron Collider* (LHC) é responsável por analisar o fluxo contínuo de colisões e selecionar, em tempo real, os eventos de interesse para armazenamento e processamento posterior. Com uma taxa de entrada de aproximadamente

40 MHz e um orçamento de latência inferior a $1\ \mu\text{s}$, essa etapa impõe restrições severas ao processamento, inviabilizando a utilização de arquiteturas computacionais convencionais [CMS Collaboration 2020]. Em termos práticos, a taxa de 40 MHz corresponde a um novo evento a cada 25 ns, e a decisão para cada evento deve ser produzida de forma determinística dentro do orçamento de latência. Como consequência, a inferência deve ser realizada de forma determinística em *hardware* dedicado, tipicamente utilizando *field-programmable gate arrays* (FPGAs), explorando sua lógica combinacional nos blocos lógicos configuráveis (CLBs) do dispositivo. A tarefa de inferência considerada neste trabalho é a classificação de jatos de partículas do LHC, representada pelo conjunto JSC [Duarte et al. 2018].

Essas restrições tornam a escolha do modelo de aprendizado de máquina tão importante quanto sua implementação em *hardware*. Entre as alternativas existentes, modelos de *Gradient Boosted Decision Trees* (GBDT) [Friedman 2001] destacam-se para tarefas de classificação em dados tabulares. Sua estrutura baseada em comparações de limiar e seleção de valores de folha pode ser mapeada de forma eficiente para primitivas LUT (*lookup table*), dispensando o uso de blocos DSP (*digital signal processing*). Além disso, GBDTs frequentemente atingem desempenho preditivo competitivo utilizando um número significativamente menor de parâmetros [Chen and Guestrin 2016, Prokhorenkova et al. 2018].

Apesar de características favoráveis à implementação em FPGA, explorar esse potencial de forma eficiente ainda é um desafio. Abordagens baseadas em HLS (*high-level synthesis*) [Summers et al. 2020, Fahim et al. 2021] normalmente convertem as árvores em circuitos sequenciais, resultando em latência proporcional à profundidade das árvores. Em uma direção diferente, o TreeLUT [Khataei and Bazargan 2025] propõe o mapeamento direto das árvores de decisão para malhas lógicas nas LUTs (lógica booleana), dispensando blocos de memória convencionais. Apesar desse avanço, a abordagem apresenta três limitações principais: (i) suporta apenas inferência *single-output*, exigindo a implementação de K *ensembles* independentes para problemas com K classes; (ii) aplica a quantização apenas após o treinamento (PTQ), restringindo a otimização conjunta do modelo e do *hardware*; e (iii) não fornece garantias formais de equivalência funcional entre o modelo treinado (em ponto flutuante) e o circuito sintetizado.

Diante dessas limitações, a questão investigada neste trabalho é se o treinamento e o *hardware* podem ser projetados de forma conjunta, de modo que o circuito sintetizado seja uma réplica exata do modelo quantizado treinado, sem perder competitividade em área e acurácia frente ao TreeLUT.

Este trabalho propõe um método restrito a árvores *simétricas* (mesmo critério de divisão em todos os nós de um nível), que aborda essas limitações por meio de cinco contribuições: (i) *forward* bit-exato durante o treinamento, em que o gradiente de *boosting* é calculado a partir de pontuações em aritmética inteira idênticos aos do *hardware* sintetizado; (ii) árvores *multi-output* com critérios de divisão compartilhados, em que uma única árvore por iteração compartilha a estrutura de divisão entre as K classes, reduzindo comparadores por um fator de até $K\times$; (iii) regularização de predicados orientada ao *hardware* (λ_{hw}), que penaliza predicados novos proporcionalmente ao custo em LUTs, reutilizando sem custo os já presentes; (iv) quantização adaptativa por iteração guiada pelo Hessiano, que ajusta a precisão fracionária de cada iteração à magnitude do gradi-

ente residual, reduzindo a largura da ROM sem retreinamento completo; e (v) otimizações de ROM pós-treinamento (*leaf shift*, *leaf merge* e *dead-leaf elimination*), que reduzem a área sem alterar o comportamento funcional. Todas as otimizações são validadas por um fluxo automatizado que exporta o modelo para VHDL sintetizável e verifica equivalência bit a bit por simulação RTL (GHDL), desde o banco de comparadores até a pontuação final de classificação.

A avaliação é conduzida nos conjuntos JSC e MNIST, com síntese e implementação no Vivado para o dispositivo `xcvu9p-flga2104-2L-e` e validação bit a bit por simulação GHDL com 10.000 vetores de teste por configuração. No JSC, o método atinge 799 LUTs com 74,56% de acurácia, em paridade praticamente exata com o TreeLUT (II) em área e acurácia; no MNIST, a Proposta III supera a acurácia do TreeLUT (II) em 0,64 pp com apenas 1,5% de área adicional. O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta o estado da arte; a Seção 3 descreve a arquitetura e o fluxo de treinamento e exportação; a Seção 4 apresenta os resultados; a Seção 5 discute os compromissos e as limitações; e a Seção 6 conclui o trabalho.

2. Estado da Arte

A implementação de redes neurais diretamente em LUTs tem sido explorada como estratégia para reduzir a latência de inferência em FPGA. Trabalhos como LogicNets [Umuroglu et al. 2020], a família PolyLUT/NeuraLUT [Andronic and Constantinides 2023, Lou et al. 2024, Andronic and Constantinides 2024, Andronic and Constantinides 2025] e DWN [Bacellar et al. 2024] representam neurônios, ou pequenas composições de neurônios, por meio de tabelas-verdade implementadas diretamente em primitivas LUT, alcançando latências inferiores a 10 ns. Essas abordagens dependem, porém, de treinamento com ativações contínuas seguido da conversão para tabelas-verdade, introduzindo uma lacuna entre o modelo treinado e o circuito sintetizado, e costumam apresentar desempenho inferior a métodos baseados em árvores de decisão em dados tabulares densos [Khataei and Bazargan 2025].

A implementação de árvores de decisão divide-se predominantemente entre fluxos HLS e arquiteturas de mapeamento espacial direto. Summers *et al.* [Summers et al. 2020] implementam árvores de decisão por HLS, representando cada árvore como uma sequência de instruções `if/else` em VHDL, com latência proporcional à profundidade. O hls4ml [Fahim et al. 2021] generaliza essa estratégia, preservando a mesma limitação de execução sequencial. Já o TreeLUT [Khataei and Bazargan 2025] converte diretamente a estrutura condicional das árvores em equações booleanas paralelizadas sintetizadas em LUTs, introduzindo registradores na árvore de soma para manter a latência de inferência constante independentemente da profundidade. Essa abordagem, no entanto, suporta apenas inferência *single-output*, aplica quantização somente pós-treinamento e não incorpora mecanismos para o controle explícito de área ou compartilhamento de partições lógicas durante o treinamento.

Diante dessas limitações, a busca por estruturas mais regulares conduz às árvores de decisão simétricas, que empregam o mesmo predicado de divisão em todos os nós de um mesmo nível, produzindo folhas de profundidade uniforme (2^d folhas para profundidade d) – restrição já explorada pelo CatBoost [Prokhorenkova et al. 2018] para re-

duzir sobreajuste. Em *hardware*, essa simetria elimina o roteamento condicional entre níveis, ou seja, o índice da folha é obtido pela simples concatenação dos resultados de comparação de cada nível. No caso especial de $d \leq 6$, a lógica pode ser mapeada em uma única primitiva LUT6 da arquitetura Xilinx, tornando essas árvores eficientes para implementações de baixa latência.

3. Metodologia

3.1. Visão Geral da Arquitetura

A arquitetura de *hardware* gerada pelo método proposto decompõe o *ensemble* GBDT em três estágios logicamente combinacionais (Figura 1): (i) um banco de comparadores compartilhado que avalia em paralelo todos os predicados distintos do *ensemble*; (ii) uma ROM de folha por iteração, endereçada por um subconjunto do vetor de comparação, que emite K valores quantizados; e (iii) uma árvore de somadores balanceada de largura progressiva que acumula as contribuições de todas as iterações por classe.

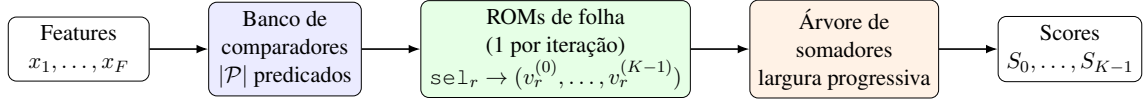


Figura 1. Arquitetura proposta: três estágios combinacionais na configuração base, um ciclo de inferência (registradores de pipeline opcionais entre estágios).

Na configuração base, os três estágios formam um único caminho combinacional, opcionalmente, registradores de *pipeline* podem ser inseridos entre estágios para atender restrições de *timing* em frequências mais altas, ao custo de latência em ciclos adicionais. O efeito da inserção desses registradores sobre área e frequência máxima é quantificado na Seção 4.

3.2. Árvores Simétricas para Hardware

Para um *ensemble* de R árvores simétricas com profundidade $d_r \leq 6$, seja $(f_{r,\ell}, b_{r,\ell})$ o critério de divisão no nível ℓ da árvore r , em que $f_{r,\ell}$ é o índice do atributo e $b_{r,\ell}$ é o índice do intervalo de quantização escolhidos. O endereço de seleção da folha da árvore r é dado por

$$\text{addr}_r = \sum_{\ell=0}^{d_r-1} \mathbb{I}[x_{f_{r,\ell}} > \theta_{f_{r,\ell}, b_{r,\ell}}] \cdot 2^{d_r-1-\ell} \quad (1)$$

em que $\theta_{f,b}$ é o limiar correspondente ao intervalo de quantização b do atributo f . Para $d = 3$, por exemplo, os três bits de comparação são concatenados diretamente, sem lógica de decisão intermediária, endereçando uma ROM de $2^3 = 8$ entradas por K classes.

No modo *multi-output*, cada iteração compartilha a estrutura de divisão entre as K classes. Nesse cenário, o conteúdo da ROM é dado por $\text{ROM}_r[\text{addr}] = (v_{r,\text{addr}}^{(0)}, \dots, v_{r,\text{addr}}^{(K-1)})$, com pontuação final $S_k = \sum_{r=0}^{R-1} \eta \cdot v_{r,\text{addr}_r}^{(k)}$. Comparado ao modo *single-output* do TreeLUT, isso reduz o número de predicados distintos por um fator de até $K \times$.

3.3. Quantização Integrada ao Treinamento (QAT)

Os valores de folha $v_{r,j}^{(k)} = -G_{r,j}^{(k)} / (H_{r,j}^{(k)} + \lambda)$ são quantizados para ponto fixo com Q_{frac} bits fracionários imediatamente após o cálculo por Newton-Raphson, e os valores quantizados $\hat{v}$ são reutilizados tanto para atualizar as previsões acumuladas quanto para calcular os gradientes da iteração seguinte, garantindo que o caminho de treinamento seja idêntico ao de inferência em *hardware*.

Para reduzir a largura da ROM, os códigos são deslocados pelo valor mínimo de cada coluna, de modo que $c_{r,j}^{(k)} = \text{round}(v_{r,j}^{(k)} \cdot 2^{Q_{\text{frac}}}) - \text{bias}_r^{(k)}$, com o viés somado como constante fora da ROM. Além disso, a precisão fracionária pode variar por iteração, guiada pelo Hessiano acumulado. Nesse caso, a precisão fracionária da iteração r é definida por

$$Q_{\text{frac},r} = \text{clip}\left(Q_{\text{frac}} + \lfloor \log_2(\bar{g}_r / g_{\text{ref}}) \rfloor, Q_{\text{min}}, Q_{\text{frac}}\right) \quad (2)$$

Iterações com maiores gradientes residuais médios podem utilizar menos bits fracionários. A penalização de custo é incorporada por λ_Q (Seção 3.6), e as pontuações são acumuladas utilizando a maior precisão fracionária empregada ao longo das iterações, $\max_r Q_{\text{frac},r}$.

3.4. Forward Bit-Exato

Durante o treinamento, a acumulação em ponto flutuante pode ser substituída por acumulação em aritmética inteira,

$$S_k^{\text{hw}} = \sum_r \text{round}\left(v_{r,\text{addr}_r}^{(k)} \cdot \eta \cdot 2^{Q_{\text{sum}}}\right), \quad (3)$$

com $Q_{\text{sum}} = \max_r Q_{\text{frac},r}$ e todas as operações realizadas em `int64`. Os gradientes da iteração seguinte são então calculados a partir de $\text{softmax}(S_k^{\text{hw}} / 2^{Q_{\text{sum}}})$, garantindo que o processo de *boosting* seja otimizado em relação ao comportamento do *hardware*.

3.5. Regularização de Predicados e Anelamento de Profundidade

O ganho de divisão padrão $\mathcal{G}(f, b)$ é penalizado pelo custo marginal de *hardware* da forma

$$\mathcal{G}_{\text{hw}}(f, b) = \mathcal{G}(f, b) - \lambda_{\text{hw}} \cdot C_{\text{novo}}(f, b) \quad (4)$$

em que $C_{\text{novo}}(f, b) = C_{\text{comp}}(f)$ se (f, b) ainda não está no banco de comparadores $\mathcal{P}_{\text{usado}}$, e 0 caso contrário. Isso incentiva a reutilização de predicados já instanciados, reduzindo comparadores sem orçamento rígido.

Adicionalmente, a profundidade máxima pode decrescer linearmente ao longo das iterações, $d_r = \max(d_{\text{min}}, d_{\text{max}} - \lfloor r \cdot \rho \rfloor)$, permitindo que iterações iniciais capturem interações de maior ordem e iterações tardias produzam correções de menor granularidade com ROMs menores.

3.6. Otimizações de ROM Pós-Treinamento

Essas otimizações são aplicadas apenas durante a exportação para VHDL, sem alterar o modelo treinado e preservando a equivalência funcional bit a bit. O σ -clip limita os valores das folhas ao intervalo $\mu \pm \sigma_{\text{clip}} \cdot \sigma$, reduzindo a quantidade de bits inteiros necessária para sua representação. O *leaf merge* agrupa folhas que compartilham o mesmo código em uma única cláusula `when`, reduzindo o número de mintermos. Por sua vez, o *dead-leaf elimination* identifica folhas nunca alcançadas durante o treinamento e as marca como `don't-care`, permitindo maior liberdade ao sintetizador durante a minimização lógica. Essa exploração de condições `don't-care` não observadas durante o treinamento para reduzir o tamanho das LUTs é consistente com resultados recentes obtidos em redes baseadas em LUT, como o ReducedLUT [Cassidy et al. 2025]. No entanto, neste trabalho, essa estratégia é aplicada diretamente à ROM de folhas de árvores simétricas, em vez de sub-tabelas associadas a neurônios.

3.7. Implementação em Hardware

O banco de comparadores avalia $\text{comp}_{f,b} = \mathbb{K}[\text{round}(x_f \cdot 2^{Q_{\text{in}}}) > \text{round}(\theta_{f,b} \cdot 2^{Q_{\text{in}}})]$ para cada $(f, b) \in \mathcal{P}$, implementado como subtrator unsigned consumindo $\lceil Q_{\text{in}}/5 \rceil$ LUTs. As ROMs de folha são instanciadas como blocos `with sel_r select`, otimizados pelo Vivado para multiplexadores LUT6. A árvore de somadores utiliza largura progressiva ($w_{\ell+1} = \max(w_{\ell}^{(a)}, w_{\ell}^{(b)}) + 1$), reduzindo a utilização de LUTs nos níveis iniciais em relação a largura uniforme.

O fluxo de exportação gera automaticamente a entidade VHDL completa (comparadores, ROMs com suporte a *leaf merge* e *dead-leaf elimination*, árvore de somadores) e um ambiente de verificação que compara, bit a bit, a saída GHDL com a saída do modelo Python, nos modos de árvore isolada e circuito completo.

3.8. Configuração Experimental

Os experimentos foram conduzidos utilizando dois *benchmarks*. O primeiro é o JSC [Duarte et al. 2018] (OpenML ID 42468), composto por aproximadamente 830.000 amostras, 16 atributos e cinco classes de jatos do LHC. O segundo *benchmark* é o MNIST [LeCun et al. 1998], contendo 70.000 imagens de dimensão 28×28 (784 atributos) distribuídas em dez classes. Ambos conjuntos de dados foram particionados em treinamento, validação e teste na proporção 80/10/10, utilizando a semente 42, com todos os atributos normalizados para o intervalo $[0, 1]$.

Os modelos foram treinados com a implementação proposta neste trabalho. Foram exploradas configurações com 10 a 80 iterações de *boosting*, profundidade máxima entre 5 e 6, 32 níveis de quantização por atributo, Q_{frac} entre 2 e 6 bits e taxa de aprendizado fixa em $\eta = 0,3$. Também foram avaliados diferentes valores para o fator de regularização em hardware, $\lambda_{\text{hw}} \in \{0, 4\}$, para o parâmetro de σ -clip (2,5–3,0) e para a fração de iterações preservadas pelo mecanismo de poda (0,6–0,8). Salvo indicação em contrário, todos os experimentos utilizam inferência em aritmética inteira durante o treinamento, suporte a múltiplas saídas e quantização de entrada com $Q_{\text{in}} = 12$ bits.

As métricas de área, frequência máxima e potência foram obtidas com o Vivado para o dispositivo `xcvu9p-flga2104-2L-e` (placa VCU118), sendo a frequência máxima derivada do *worst negative slack* pós-implementação (Seção 4.2) e a potência

estimada após o roteamento (Seção 4.3). A equivalência funcional foi verificada por simulação RTL com o GHDL. Os resultados não incluem execução em placa FPGA.

4. Resultados

Esta seção avalia o método proposto, analisando o impacto de seus componentes e sua generalização para o MNIST, bem como o efeito da inserção de registradores sobre frequência e área (Seção 4.2) e a comparação direta com o TreeLUT no JSC (Seção 4.4).

4.1. Impacto dos Componentes

A Tabela 1 resume quatro análises controladas de componentes (mesma base de hiperparâmetros dentro de cada bloco).

Tabela 1. Impacto dos componentes no JSC (profundidade fixa em 6; *em λ_Q , Q_{frac} atua como teto, não valor fixo).

Componente	Configuração	Iterações	Q_{frac}	λ_{hw}	Keep	Q_{min}	Predicados	Acc (%)	LUTs
Tipo de saída	Single-output	50	6	4,0	—	—	333	75,59	4.160
	Multi-output	50	6	4,0	—	—	188	75,22	3.867
λ_{hw}	0,0	50	4	0,0	0,7	—	141	75,01	1.812
	4,0	50	4	4,0	0,7	—	142	75,03	1.775
Q_{frac}	4	50	4	4,0	0,7	—	142	75,03	1.775
	5	50	5	4,0	0,7	—	146	74,98	2.266
	6	50	6	4,0	0,7	—	139	74,94	2.508
Otim. ROM	baseline (σ -clip)	50	4	4,0	0,7	—	—	75,03	1.913
	+ leaf merge	50	4	4,0	0,7	—	—	75,03	1.883
	+ dead-leaf elim.	50	4	4,0	0,7	—	—	75,03	1.775
λ_Q	ausente	30	4	0,0	0,8	2	—	74,75	1.357
	9×10^{-3}	30	4*	0,0	0,8	2	—	74,67	1.247
	$2,4 \times 10^{-2}$	30	4*	0,0	0,8	2	—	74,59	1.276
	7×10^{-2}	30	4*	0,0	0,8	2	—	74,59	1.263

A comparação entre as arquiteturas *multi-output* e *single-output* evidencia um compromisso clássico entre área e desempenho. A representação *multi-output* reduz o número de predicados em 43,5% (333→188), resultando em uma diminuição de 7,0% na área, ao custo de uma redução de 0,37 pp na acurácia. Dessa forma, a abordagem não representa uma melhoria simultânea em ambos os critérios, mas sim um ponto distinto da fronteira de Pareto.

Já a introdução da regularização por *hardware* ($\lambda_{\text{hw}} = 4$) produz um comportamento mais favorável, uma vez que reduz a área em aproximadamente 2,0% e melhora o caminho crítico sem impacto mensurável na qualidade do modelo. A acurácia permanece praticamente inalterada, indicando que essa regularização consegue eliminar redundâncias da arquitetura preservando o desempenho preditivo.

Resultados semelhantes são observados na análise da precisão fracionária. A redução de Q_{frac} de 6 para 4 bits proporciona uma diminuição de 29,2% na área sem perda de acurácia. Esse resultado evidencia que, para o problema considerado, a precisão originalmente utilizada era superior ao necessário para representar adequadamente os limiares do modelo.

As otimizações aplicadas às ROMs apresentam contribuições distintas. Apesar do *leaf merge* identificar uma taxa de colisão de folhas de 50,9%, seu impacto isolado na área é limitado a 1,6%, indicando que o sintetizador já elimina parte dessa redundância automaticamente. Por outro lado, a eliminação de folhas mortas (*dead-leaf elimination*) introduz condições de *don't-care* que não seriam inferidas pelo sintetizador, produzindo um ganho acumulado de 5,7%. Também foi investigada a deduplicação de ROMs idênticas entre iterações, porém, nenhuma colisão exata foi observada em qualquer configuração avaliada. Como a comparação considera simultaneamente $n_{\text{leaves}} \times K$ valores, a probabilidade de coincidência exata torna-se extremamente baixa mesmo em precisões menores, tornando essa otimização ineficaz no contexto estudado.

A regularização proposta por meio de λ_Q também promove redução de área, tipicamente entre 6% e 8%, ao custo de até 0,16 pp de acurácia. Porém, o comportamento observado não é estritamente monotônico, uma vez que a menor intensidade de penalização já produz a menor área obtida, sugerindo que o ganho satura rapidamente após o colapso das iterações com menor contribuição para o gradiente. O efeito torna-se mais evidente quando essa estratégia é combinada com a regularização por *hardware* no experimento com MNIST, apresentado na Seção 4.4. Por fim, como validação funcional da metodologia proposta, todas as configurações foram exportadas para VHDL e comparadas com simulações RTL (GHDL) utilizando 10.000 vetores de teste por configuração. Em todos os casos foi obtida correspondência bit a bit de 100,0% entre as saídas da implementação de referência em Python e da implementação em *hardware*.

4.2. Impacto de Registradores na Frequência Máxima e Área

Para avaliar o custo real de inserir registradores entre os três estágios combinacionais, seis configurações foram implementadas e comparadas à referência combinacional sob o mesmo período alvo (5,0 ns, 200 MHz). A frequência máxima efetiva é derivada do *worst negative slack* pós-implementação, $F_{\text{max}} = 1/(T_{\text{clk}} - \text{WNS})$. A Tabela 2 resume os resultados.

Tabela 2. Efeito do *pipeline* em LUTs e frequência máxima pós-implementação.

Config	LUTs comb.	Fmax comb. (MHz)	LUTs pipe	Fmax pipe (MHz)	Δ LUT	Ganho Fmax
Proposta I (JSC)	564	290,7	593	306,4	+5,1%	+5,4%
Proposta II (JSC)	799	273,7	783	342,9	−2,0%	+25,3%
Proposta III (JSC)	1.278	241,8	1.148	482,4	−10,2%	+99,5%
Proposta IV (JSC)	1.775	231,6	1.762	613,9	−0,7%	+165,1%
Proposta V (MNIST)	8.603	203,3	7.666	476,6	−10,9%	+134,5%

O orçamento de latência do gatilho de Nível-1 do LHC (Seção 1) opera na faixa de dezenas a poucas centenas de MHz. Mesmo blocos de processamento mais exigentes implementados em FPGA, como o *L1 Track Finder* do CMS para o HL-LHC, também implementado em Xilinx UltraScale+, operam a 240 MHz [Yates and CMS Collaboration 2021]. A frequência máxima obtida neste trabalho, mesmo para a implementação puramente combinacional, excede esse requisito em todas as configurações avaliadas. Dessa forma, uma comparação direta com trabalhos cujo objetivo é minimizar o período de clock, como o TreeLUT (1,0–1,5 ns, correspondendo a aproximadamente 660 MHz–1 GHz [Khataei and Bazargan 2025]), não é a mais apropriada, pois ambos os métodos foram desenvolvidos considerando restrições de projeto

distintas. No contexto da aplicação considerada neste trabalho, a frequência alcançada já apresenta ampla margem em relação aos requisitos de operação do gatilho de Nível-1, de modo que ganhos adicionais de frequência deixam de representar um benefício prático.

Apesar da expectativa de uma relação simples entre latência e frequência máxima, a inserção de registradores reduz a área em cinco das seis configurações, ao mesmo tempo em que aumenta a frequência máxima, chegando a $-10,9\%$ no caso MNIST de maior profundidade. O ganho de frequência cresce com a profundidade lógica do caminho crítico, passando de $+5,4\%$ na Proposta I (profundidade efetiva 2) para $+165,1\%$ na Proposta IV (10 estágios). O uso de CARRY8 aumenta de 0–18 (implementação combinacional) para 42–663 (*pipeline*) nos casos de maior ganho, sugerindo melhor aproveitamento da *carry-chain* ao isolar a árvore de somadores em um estágio dedicado. A exceção é a Proposta I, pois a profundidade lógica já é mínima e o *pipeline* não dispõe de lógica suficiente para reorganizar a implementação, resultando em um leve aumento de área ($+5,1\%$).

4.3. Análise de Potência

Como o TreeLUT não reporta resultados de potência em nenhuma de suas variantes, não é possível realizar uma comparação direta sob esse aspecto. Ainda assim, os resultados apresentados estabelecem um ponto de referência para avaliações de potência em arquiteturas de mapeamento GBDT para LUT. A potência foi estimada após o roteamento utilizando o comando `report_power` do Vivado, no dispositivo `xcvu9p-flga2104-2L-e` (placa VCU118), com atividade de comutação obtida por meio de simulação bit-exata utilizando arquivos SAIF (*Switching Activity Interchange Format*), abrangendo todas as implementações combinacionais apresentadas na Tabela 3.

Tabela 3. Potência pós-rota por ponto de Pareto (Vivado `report_power`, XCVU9P/VCU118, 25 °C, típico). Dinâmica reflete comutação do design; Estática é o *leakage* do dispositivo, quase constante entre configurações. SAIF: cobertura de nets por atividade real de simulação; Confiança: Medium em todos os pontos.

Dataset	Config	LUTs	Dinâmica (mW)	Estática (W)	Total (W)	SAIF (%)
JSC	Proposta I	564	53	2,945	2,998	21
	Proposta II	799	96	2,946	3,042	17
	Proposta III	1.278	409	2,952	3,361	10
	Proposta IV	1.775	626	2,956	3,582	10
MNIST	Proposta I	2.613	525	2,954	3,479	35
	Proposta II	2.735	531	2,954	3,485	39
	Proposta III	3.550	1.094	2,965	4,058	37
	Proposta IV	4.495	1.575	2,974	4,549	32
	Proposta V	8.603	3.973	3,021	6,994	25

Observa-se que a potência dinâmica aumenta monotonicamente com o número de LUTs utilizadas, variando de 53 mW (564 LUTs) a 626 mW (1.775 LUTs) para o JSC e de 525 mW a 3.973 mW (2.613–8.603 LUTs) para o MNIST. Esse comportamento é consistente com o aumento da atividade de comutação associado ao crescimento da lógica combinacional. Em contrapartida, a potência estática permanece praticamente constante, entre aproximadamente 2,95 W e 3,02 W, por refletir predominantemente a corrente de fuga do dispositivo XCVU9P como um todo, e não apenas do circuito implementado. Dessa forma, a potência dinâmica constitui a métrica mais representativa para comparações entre diferentes implementações.

Em todas as configurações avaliadas, o Vivado reporta nível de confiança *Medium*, com cobertura de SAIF entre 10% e 39%. A atividade dos sinais não cobertos é estimada pelo mecanismo *vectorless* da ferramenta, uma vez que parte das redes geradas após a síntese não possui correspondência direta com os sinais nomeados no RTL original. Assim, os valores de potência dinâmica devem ser interpretados como estimativas baseadas em atividade de comutação parcialmente observada, e não como medições experimentais. Além disso, não foi estimada uma barra de erro para essas métricas, de modo que os valores absolutos não devem ser interpretados como medição; sua utilidade está na comparação relativa entre configurações, obtidas com a mesma ferramenta, dispositivo e metodologia.

4.4. Comparação com o TreeLUT

A Tabela 4 compara o método proposto ao TreeLUT nos dois *benchmarks*.

Tabela 4. Comparação entre o método proposto e o TreeLUT no JSC e no MNIST.

Dataset	Modelo	Iterações	Profundidade	Q_{frac}	Acc (%)	AUC	LUTs
JSC	TreeLUT (II)	10	5	2	74,6	—	796
	TreeLUT (I)	13	5	4	75,6	—	2.234
	Proposta I (anneal)	30	6→2	2	74,12	0,9278	564
	Proposta II	30	6	2	74,56	0,9295	799
	Proposta III	50	5	3	74,75	0,9325	1.278
	Proposta IV	50	6	4	75,03	0,9333	1.775
MNIST	TreeLUT (II)	—	6	—	95,6	—	3.499
	TreeLUT (I)	—	6	—	96,6	—	4.478
	Proposta I (compacto)	30	6	4	94,40	—	2.613
	Proposta II (keep 0,6)	50	6	3	95,51	—	2.735
	Proposta III	50	6	3	96,24	—	3.550
	Proposta IV (HQAT+ λ_{hw})	50	6	4	96,44	—	4.495
	Proposta V (80r)	80	6	5	97,09	—	8.603

Cabe ressaltar que as configurações da Tabela 4 diferem em número de iterações, profundidade e precisão fracionária. Por esse motivo, a comparação é feita sobre a fronteira de Pareto entre área e acurácia, e não entre configurações com hiperparâmetros equivalentes. O TreeLUT é adotado como principal referência quantitativa por ser o trabalho mais próximo, ao mapear as árvores de decisão diretamente em lógica de LUTs, enquanto as abordagens baseadas em HLS mantêm execução sequencial e as redes neurais em LUT adotam outro paradigma de modelo, sendo discutidas na Seção 2. A Tabela 1 isola o efeito de cada mecanismo no JSC sobre uma mesma base, enquanto as configurações da Tabela 4 combinam esses mecanismos; o efeito combinado de λ_{hw} e da quantização guiada pelo Hessiano no MNIST é detalhado a seguir.

No JSC, a Proposta II atinge paridade praticamente exata com o TreeLUT (II) em área e em acurácia. Proposta I reduz a área em 29% adicionais ao custo de 0,44 pp, estabelecendo o piso da fronteira observada. No extremo de maior acurácia, Proposta IV permanece 20,5% menor que o TreeLUT (I), com 0,57 pp menos de acurácia.

No MNIST, a Proposta III supera o TreeLUT (II) em acurácia por 0,64 pp com apenas 1,5% de área adicional, aproximando-se de uma situação de dominância estrita. Além disso, a Proposta IV demonstra que a introdução da penalização por *hardware* ($\lambda_{\text{hw}} = 4$) continua sendo efetiva mesmo quando a quantização das folhas já foi aplicada. Em relação à configuração equivalente sem essa penalização, a Proposta IV reduz a área

de 4.550 para 4.495 LUTs e aumenta a acurácia de 96,10% para 96,44%, evidenciando ganhos simultâneos em ambos os critérios.

5. Discussão e Limitações

O compartilhamento de critérios de divisão entre classes constitui a principal vantagem arquitetural do método proposto sobre o TreeLUT, pois reduz proporcionalmente o número de comparadores e a latência de roteamento em tarefas com $K > 2$ classes, sendo esse benefício mais expressivo em conjuntos de dados com muitos atributos. O *forward* bit-exato transforma a correspondência treino-*hardware* em um invariante estrutural, não uma propriedade verificada *a posteriori*, particularmente importante no gatilho de Nível-1 do LHC, em que qualquer divergência pode levar a taxas de gatilho fora da especificação.

O método proposto oferece múltiplos mecanismos independentes para controlar a relação entre área e acurácia, incluindo λ_{hw} (predicados), Q_{frac}/λ_Q (largura da ROM), o anelamento de profundidade e otimizações pós-treinamento que comprimem a ROM sem alterar sua funcionalidade. A combinação desses mecanismos permite obter implementações entre aproximadamente 560 e 1.800 LUTs no JSC, mantendo acurácia competitiva com o TreeLUT em toda essa faixa de área.

Nem todas as otimizações inicialmente propostas se mostraram eficazes. A deduplicação de ROMs entre iterações apresentou taxa de colisão exata nula em todos os cenários avaliados (Seção 4) e, por esse motivo, foi descartada e não foi incluída entre as contribuições do trabalho.

A restrição de simetria reduz a expressividade de cada árvore em comparação com árvores não simétricas. Esse efeito é parcialmente compensado pela possibilidade de utilizar um maior número de iterações, viabilizado pela menor área ocupada por árvore. Uma indicação do compromisso resultante é dada pela comparação com o TreeLUT (Tabela 4): no JSC, a Proposta IV apresenta 0,57 pp a menos de acurácia que o TreeLUT (I), com área 20,5% menor, e, no MNIST, a Proposta IV apresenta 0,16 pp a menos de acurácia com área equivalente (4.495 contra 4.478 LUTs). Essas diferenças não isolam o efeito da simetria, uma vez que as configurações também diferem em quantização, profundidade e número de iterações. Um estudo controlado com árvores não simétricas no mesmo fluxo de treinamento é deixado como trabalho futuro.

Na implementação combinacional, a latência é determinada pela profundidade lógica formada pelos comparadores, ROM e árvore de somadores, correspondendo tipicamente a 4–8 níveis de LUT6 ($\sim 1,6$ – $3,2$ ns, desconsiderando o atraso de roteamento). As variantes com registradores aumentam a frequência máxima de operação e, na maioria dos casos avaliados, também reduzem a área, ao custo da introdução de estágios adicionais de registradores (Seção 4.2).

Os resultados apresentados possuem limitações. Primeiro, a avaliação restringe-se aos conjuntos JSC e MNIST, nos quais o TreeLUT reporta resultados para comparação (Tabela 4), sendo necessárias avaliações em outros conjuntos tabulares multiclasse para verificar a generalidade do método. Segundo, os resultados de área, frequência e potência foram obtidos por síntese e implementação no Vivado, com equivalência funcional verificada por simulação RTL, mas sem execução em placa FPGA. Por fim, a análise de potência (Seção 4.3) foi realizada apenas para as implementações combinacionais, com

nível de confiança *Medium*, cobertura de SAIF entre 10% e 39% e sem estimativa de erro, caracterizando uma estimativa baseada em simulação, e não uma medição experimental.

6. Conclusões

Este trabalho apresentou um método para o mapeamento direto de *ensembles* GBDT com árvores simétricas para primitivas LUT de FPGA, no qual a correspondência exata entre treinamento e *hardware* é preservada como um invariante estrutural ao longo de todo o processo de *boosting*. As principais contribuições incluem o *forward* bit-exato, árvores *multi-output* com critérios de divisão compartilhados, regularização de predicados orientada ao *hardware*, quantização adaptativa por iteração guiada pelo Hessiano e otimizações pós-treinamento aplicadas à ROM (*leaf shift*, *leaf merge* e *dead-leaf elimination*). Os resultados experimentais demonstram a eficácia da abordagem proposta. No JSC, a Proposta II iguala em área e acurácia a TreeLUT (II), enquanto, no MNIST, a Proposta III supera a acurácia do TreeLUT (II) com área praticamente equivalente. Além disso, todas as implementações são validadas por um fluxo automatizado de geração de VHDL e verificação bit a bit utilizando GHDL, assegurando equivalência funcional entre o modelo treinado e a implementação em *hardware*. Como perspectivas para trabalhos futuros, destacam-se a utilização de destilação de conhecimento para obtenção de *ensembles* mais compactos, a exploração automática das configurações de *hardware* para diferentes compromissos entre área e acurácia, a validação em placa FPGA, a avaliação em outros conjuntos de dados tabulares multiclasse, a comparação controlada com árvores não simétricas e a caracterização de potência das variantes com registradores (Seção 4.2), complementando a análise das implementações combinacionais apresentada na Seção 4.3.

Referências

- Andronic, M. and Constantinides, G. A. (2023). Polylut: Learning piecewise polynomials for ultra-low latency fpga lut-based inference. In *International Conference on Field Programmable Technology (ICFPT)*, pages 60–68.
- Andronic, M. and Constantinides, G. A. (2024). Neuralut: Hiding neural network density in boolean synthesizable functions. In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 140–148.
- Andronic, M. and Constantinides, G. A. (2025). Neuralut-assemble: Hardware-aware assembling of sub-neural networks for efficient lut inference. *arXiv preprint arXiv:2504.00592*.
- Bacellar, A. T. L., Susskind, Z., Breternitz Jr, M., John, E., John, L. K., Lima, P. M. V., and França, F. M. G. (2024). Differentiable weightless neural networks. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235, pages 2277–2295.
- Cassidy, O., Andronic, M., Coward, S., and Constantinides, G. A. (2025). Reduce-dLUT: Table decomposition with “don’t care” conditions. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA ’25)*. arXiv:2412.18579.
- Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.

- CMS Collaboration (2020). The phase-2 upgrade of the cms level-1 trigger. Technical Report CERN-LHCC-2020-004, CERN, Geneva.
- Duarte, J. et al. (2018). Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation*, 13:P07027. Introduces the hls4ml_lhc_jets_hlf (JSC) dataset, OpenML ID 42468.
- Fahim, F. et al. (2021). hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv preprint arXiv:2103.05579*.
- Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232.
- Khataei, A. and Bazargan, K. (2025). TreeLUT: An efficient alternative to deep neural networks for inference acceleration using gradient boosted decision trees. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '25)*, pages 1–11. arXiv:2501.01511.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Lou, B., Rademacher, R., Boland, D., and Leong, P. H. W. (2024). Polylut-add: Fpga-based lut inference with wide inputs. In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 149–155.
- Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2018). CatBoost: Unbiased boosting with categorical features. *Advances in Neural Information Processing Systems (NeurIPS)*, 31.
- Summers, S. et al. (2020). Fast inference of boosted decision trees in fpgas for particle physics. *Journal of Instrumentation*, 15:P05026.
- Umuroglu, Y., Akhauri, Y., Fraser, N. J., and Blott, M. (2020). Logicnets: Co-designed neural networks and circuits for extreme-throughput applications. In *International Conference on Field-Programmable Logic and Applications (FPL)*.
- Yates, B. R. and CMS Collaboration (2021). A cms level-1 track finder for the hl-lhc. *arXiv preprint arXiv:2110.02826*.