

Universidade Federal de Santa Catarina  
Campus Reitor João David Ferreira Lima  
Departamento de Engenharia Elétrica e Eletrônica



Luis Antonio Spader Simon

Geração automatizada de descrição VHDL de aceleradores de  
Redes Neurais Artificiais

Florianópolis  
2023

**Luis Antonio Spader Simon**

# **Geração automatizada de descrição VHDL de aceleradores de Redes Neurais Artificiais**

Ante-projeto de Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos necessários para a obtenção do título de Bacharel em Engenharia Eletrônica.

Orientador: Prof. Dr. Héctor Pettenghi

Universidade Federal de Santa Catarina  
Campus Reitor João David Ferreira Lima  
Departamento de Engenharia Elétrica e Eletrônica

Florianópolis  
2023

**Luis Antonio Spader Simon**

# **Geração automatizada de descrição VHDL de aceleradores de Redes Neurais Artificiais**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Bacharel em Engenharia Eletrônica” e aprovado em sua forma final pelo Curso de Graduação em Engenharia Eletrônica.

Florianópolis, 9 de outubro de 2023

---

Prof. Fernando Rangel de Souza, Dr.  
Coordenador do Curso

## **Banca Examinadora:**

---

Prof. Dr. Héctor Pettenghi  
Universidade Federal de Santa Catarina  
Orientador

---

Prof. Dr. Mateus Grellert  
Universidade Federal de Santa Catarina

---

Prof. Dr. Fabian L. Cabrera  
Universidade Federal de Santa Catarina

---

Prof. Dr. José Luiz Almada Güntzel  
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus familiares, namorada e amigos, os quais foram os pilares desta jornada. Sem vocês isso não seria possível.

# Agradecimentos

Gostaria de expressar meus sinceros agradecimentos a todas as pessoas que desempenharam papéis fundamentais na minha jornada acadêmica e na realização deste trabalho:

Aos meus pais, Nilssei e Donizete, que não apenas me apoiaram financeiramente, mas também me proporcionaram apoio emocional ao longo desses anos na UFSC.

À minha namorada, Milena, que esteve ao meu lado, oferecendo orientação, incentivo e persistência durante todo o processo de conclusão deste trabalho.

Aos meus professores orientadores, Mateus Grellert e Héctor Pettenghi, que investiram tempo e esforço significativos para me auxiliar na elaboração e desenvolvimento deste trabalho.

Aos professores do laboratório ECL, José Luiz Güntzel e Cristina Meinhardt, por fornecerem o suporte e a estrutura necessários para a realização deste projeto. Aos meus colegas de curso e os colegas do laboratório ECL, os quais me proporcionaram companhia e bons momentos que me ajudaram a suportar os desafios durante esta jornada de graduação.

# Resumo

Algoritmos de Inteligência Artificial (IA) estão em tendência crescente de uso devido ao potencial em soluções no mercado. Porém o grande uso destes algoritmos em arquiteturas de hardware não preparadas para este tipo de execução deixa uma grande pegada de carbono no planeta. Diversos estudos demonstram que o uso de arquiteturas dedicadas fornecem um notável aumento de eficiência energética, poder de processamento e diminuição no consumo.

Porém o projeto de arquiteturas dedicadas requer tempo e demonstra não conseguir acompanhar a contínua evolução dos algoritmos de IA. Logo, a proposta deste trabalho se baseia na construção de um framework para geração automática de descrição de hardware em linguagem VHDL com objetivo de automatizar síntese em hardware das diferentes arquiteturas através de configurações parametrizáveis e modelos de entrada. Acelerar o projeto de arquiteturas com o *framework* traz 2 possibilidades: a rápida síntese em FPGAs para uso final ou prototipação e a rápida síntese em ASIC; Os benefícios disto trazem a diminuição dos custos e tempo de projeto e uma maior proximidade de arquiteturas sintetizadas com os modelos recentemente treinados.

Ao final apresenta-se o resultado e estudo comparativo das diferentes arquiteturas possíveis a serem geradas.

**Palavras-Chave:** 1. Redes Neurais Artificiais. 2. Síntese de hardware de alto nível (HLS). 3. Hardware.

# Listas de figuras

Figura 1 – Poder computacional usado no treinamento de sistemas de IA.	13
Figura 2 – Desempenho/Watt Relativo (TPU, CPU e GPU).	13
Figura 3 – Aumento de velocidade de Processamento.	14
Figura 4 – Aumento de eficiência energética (HAN et al., 2016).	14
Figura 5 – Comparação de desempenho entre arquitetura dedicada <i>Descartes</i> e CPU, GPU.	15
Figura 6 – Comparação de desempenho entre diversos aceleradores.	15
Figura 7 – Lei de Moore e seu fim.	16
Figura 8 – Fluxo estimado de trabalho.	17
Figura 9 – Classificação de algoritmos de aprendizado de máquina.	21
Figura 10 – Arquitetura de uma Rede Neural Artificial genérica (ANN $i \rightarrow h_1 \rightarrow h_2 \rightarrow h_n \rightarrow o$ ).	22
Figura 11 – Diagrama de blocos do Perceptron.	23
Figura 12 – Exemplo de Funções de Ativação.	23
Figura 13 – Arquitetura genérica de um <i>AutoEncoder</i> .	24
Figura 14 – Arquitetura DPUCZDX8G para placas da categoria Zynq UltraScale+	26
Figura 15 – Fluxo de trabalho do <i>framework</i> hls4ml.	27
Figura 16 – Arquiteturas de multiplicação combinacional (pesos variáveis vs pesos fixos).	30
Figura 17 – Blocos de soma e deslocamentos (Spiral).	32
Figura 18 – Substituição das somas por um bloco de compressores.	33
Figura 19 – <i>Carry-Save Adders</i> .	33
Figura 20 – Arquitetura genérica de uma Rede Neural Artificial de Perceptrons multi-camadas (MLPs) gerada pelo framework.	34
Figura 21 – Funções de ativação disponíveis	35
Figura 22 – Exemplo de uma arquitetura em hardware de um MLP gerado pelo framework proposto.	36
Figura 23 – Shift-register structure.	37
Figura 24 – Arquitetura genérica de um Neurônio (exemplo para arquitetura com 4 entradas).	38
Figura 25 – Fluxograma do framework	38
Figura 26 – Resultados de síntese para a frequência alvo de 200 MHz.	40

- Figura 27 – Desempenho na inferência de classificação considerando a variação do número total de camadas (N) e bits de precisão (B) (imagem 4x4). . . 42
- Figura 28 – Desempenho na inferência de classificação considerando a variação do número total de camadas (N) e bits de precisão (B) (imagem 8x8). . . 43

# Sumário

1	INTRODUÇÃO . . . . .	11
1.1	Problema 1: Poder de Processamento e Consumo Energético	12
1.2	Problema 2: Projeto de arquiteturas dedicadas . . . . .	16
1.3	Metodologia . . . . .	16
1.4	Motivação e Objetivos . . . . .	18
1.4.1	Motivação . . . . .	18
1.4.2	Objetivos Gerais . . . . .	19
1.4.3	Objetivos Específicos . . . . .	19
2	REFERENCIAL TEÓRICO . . . . .	20
2.1	<i>Machine Learning</i> . . . . .	20
2.2	<i>Deep Learning</i> . . . . .	21
2.2.1	Perceptron . . . . .	22
2.2.2	Funções de Ativação . . . . .	22
2.2.3	Autoencoders . . . . .	23
3	TRABALHOS RELACIONADOS . . . . .	25
3.0.1	Vitis AI . . . . .	25
3.1	Ferramentas de código-aberto . . . . .	26
3.1.1	hls4ml . . . . .	26
3.1.2	NNGen . . . . .	27
3.2	Tabela Comparativa dos principais <i>Frameworks</i> . . . . .	28
4	PROPOSTA . . . . .	29
4.1	Framework . . . . .	29
4.2	Melhorias para Arquiteturas Geradas pelo Framework . . . . .	29
4.2.1	Substituição dos Multiplicadores por Blocos de Somas e Deslocamentos . . . . .	31
4.2.2	Troca dos Somadores por um Bloco de Compressão de Vetores	32
5	RESULTADOS . . . . .	34
5.1	Framework . . . . .	34
5.1.1	Personalização da Largura de Representação de Bits . . . . .	34

5.1.2	Escolha de Funções de Ativação . . . . .	35
5.1.3	Inclusão de Barreiras de Registradores . . . . .	35
5.1.4	Modos de Atualização de Pesos e Inferência . . . . .	36
5.1.5	Modularização das Arquiteturas . . . . .	37
5.1.6	Resultados e Limitações . . . . .	38
5.2	Resultados de Síntese . . . . .	39
5.2.1	Resultados de Síntese . . . . .	39
5.2.2	Efeitos da Aproximação em Redes Neurais de Classificação . . . . .	41
5.3	Melhorias para arquiteturas geradas pelo <i>Framework</i> . . . . .	43
5.3.1	Substituição dos Multiplicadores por Blocos de Somas e Deslocamentos . . . . .	44
5.3.2	Troca dos Somadores por um Bloco de Compressão de Vetores	46
6	CONCLUSÃO . . . . .	48
	REFERÊNCIAS BIBLIOGRÁFICAS . . . . .	50
	APÊNDICE A – TABELA COMPARATIVA DE FRAMEWORKS	56

# 1 Introdução

Apesar de algoritmos de inteligência artificial terem começado seu estudo no ano 1956 no campus de Dartmouth College (USA), apenas recentemente, a partir dos anos 2000 os algoritmos de inteligência artificial (IA) vêm se popularizando. Isto se deu ao fato do aprimoramento dos algoritmos com o surgimento dos sistemas especialistas (*expert systems*) a partir dos anos 80 (WIKIPEDIA, 2022b), aumento do poder de processamento dos computadores seguindo a Lei de Moore (WIKIPEDIA, 2022b) com o uso das GPUs<sup>1</sup> (SCHMIDHUBER, 2015) e atual demanda por diversas utilizações na indústria (PAN, 2016).

O mercado para este setor demonstra uma tendência crescente onde até o ano 2030, estima-se que o setor estará avaliado em 1.597,1 bilhões de dólares (PRECEDENCERESEARCH, 2022).

Devido ao aprimoramento dos algoritmos, atualmente encontram-se diversas aplicações para a utilização dos mesmos como por exemplo a detecção à fraude em bancos (DESROUSSEAUX; BERNARD; MARIAGE, 2021) em tempo real, através de algoritmos especializados na detecção de anomalias, *Video Coding & Decoding* (ZHOU; LV; YI, 2022) e (ZHANG et al., 2019) onde algoritmos de Inteligência Artificial (IA) podem ser utilizados para substituir ou melhorar parte do processo de codificação (LU et al., 2021) ou ainda toda a codificação e decodificação (PESSOA et al., 2020).

Se tratando dos modelos de algoritmo de IA, modelos de redes convolucionais (CNNs) e redes neurais artificiais recorrentes (RNNs) têm sido propostos para imagem, vídeo a processamento de fala (GUO et al., 2019). CNNs 1D foram propostas recentemente e imediatamente alcançaram níveis de desempenho de ponta em várias aplicações, como classificação de dados biomédicos personalizados e diagnóstico precoce, monitoramento de integridade estrutural, detecção e identificação de anomalias em eletrônica de potência e elétrica detecção de falha do motor (K et al., 2021).

Outra aplicação de IA é na orientação de veículos autônomos usando principalmente algoritmos de detecção de objetos (OYANG et al., 2020), planejadores de caminhos locais e sentido de rotas (ISELE et al., 2018) e (WANG et al., 2020), onde pretende-se

---

<sup>1</sup>**GPU (Graphical Processing Units):** Unidade de processamento gráfico, um processador especializado originalmente projetado para acelerar a renderização de gráficos. Projetada para processamento paralelo, a GPU é usada em uma ampla gama de aplicações, incluindo gráficos e renderização de vídeo. Embora sejam mais conhecidas por suas capacidades em jogos, as GPUs estão se tornando mais populares para uso em produção criativa e inteligência artificial (IA).

futuramente a associação sistemas de completos de direção autônoma, porém atualmente a principal dificuldade é a falta de algoritmos de segurança desenvolvidos de forma determinística e da dificuldade em se interpretar a segurança dos algoritmos diante das infinitas possibilidades de ocasiões (YURTSEVER et al., 2020).

## 1.1 Problema 1: Poder de Processamento e Consumo Energético

Conforme comentado anteriormente, o poder de processamento foi um dos fatores que permitiu a utilização de algoritmos de IA. O poder de processamento atual é em grande parte responsabilizado pelo uso das GPUs, que são utilizadas tanto para treinamento dos algoritmos, (que não construídos deterministicamente, mas sim passam por um ciclo iterativo até atingir o desempenho desejado para cada aplicação) quanto para inferência. O problema das GPUs é sua eficiência energética e o consumo de energia em algoritmos de IA está deixando marcas no planeta. Um trabalho de pesquisa da Universidade de Massachusetts Amherst (STRUBELL; GANESH; MCCALLUM, 2019) estimou que, durante uma vida útil média, as emissões de carbono associadas ao treinamento de um modelo de transformador, como BERT ou GPT-2, com pesquisa de arquitetura neural são equivalentes à pegada de carbono de um carro, incluindo combustível (MORGAN, 2021).

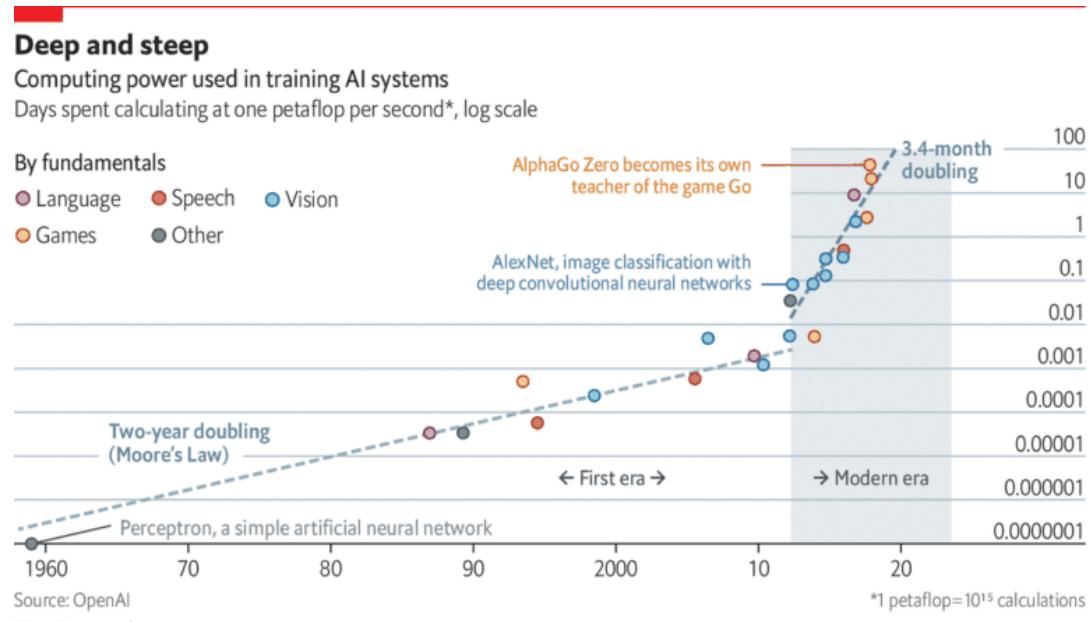
Para implementar uma DNN de alta eficiência e baixo consumo energético, pode-se otimizar em 2 frentes: no design e algoritmo do modelo DNN ou na aceleração em hardware (HAO; CHEN, 2018).

Apesar do alto poder de processamento das GPUs, como as mesmas não foram construídas especificamente para algoritmos de IA, arquiteturas dedicadas conseguem trazer uma eficiência e poder de processamento maiores (JOUSSI, 2017), (NURVITADHI et al., 2016) e (HAN et al., 2016).

Han et al. (2016) desenvolveram arquiteturas dedicadas que conseguem alcançar uma eficiência energética até 1052 vezes maior do que GPU (NVIDIA GeForce GTX Titan X) e 24.207 vezes maior que CPU (Intel Core i-7 5930k) e simultaneamente o processamento superior em até 4 vezes maior que a GPU citada anteriormente (HAN et al., 2016).

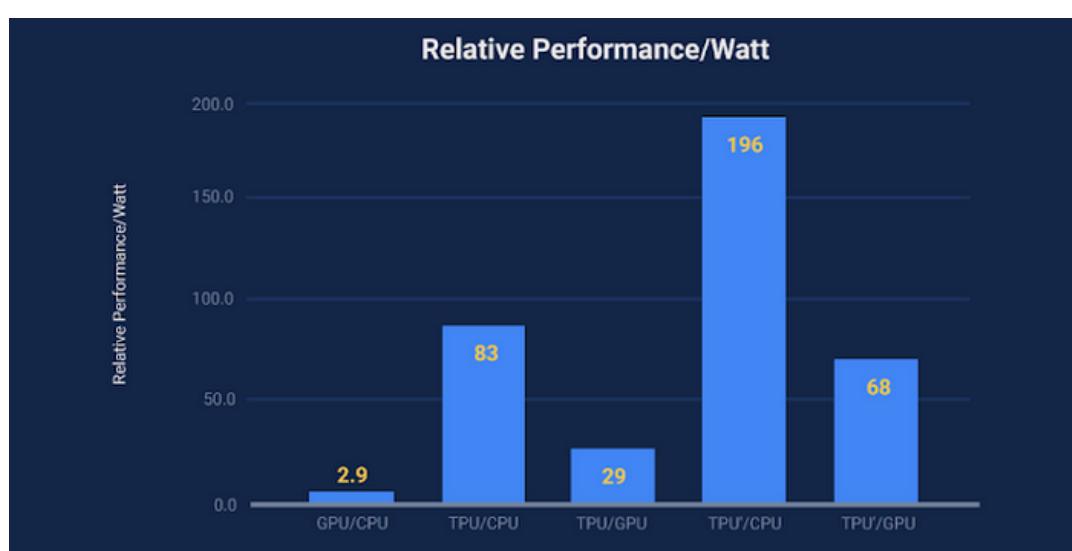
Recentemente a empresa Google optou por desenvolver arquiteturas próprias para uso em *Machine Learning* (ML), ao custo de dezenas de milhões de dólares. Apesar de todo este custo, (Joussi et al., 2017) (THOMPSON; SPANUTH, 2018) afirma terem avançado

Figura 1 – Poder computacional usado no treinamento de sistemas de IA.



Fonte: (LAI SUBTAI AHMAD; MAVER, 2022).

Figura 2 – Desempenho/Watt Relativo (TPU, CPU e GPU).

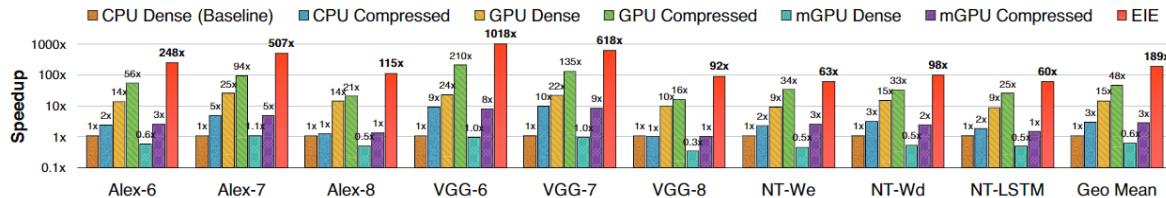


Fonte: (JOUUPPI, 2017).

em poder de processamento o equivalente a 7 anos da Lei de Moore (THOMPSON; SPANUTH, 2018).

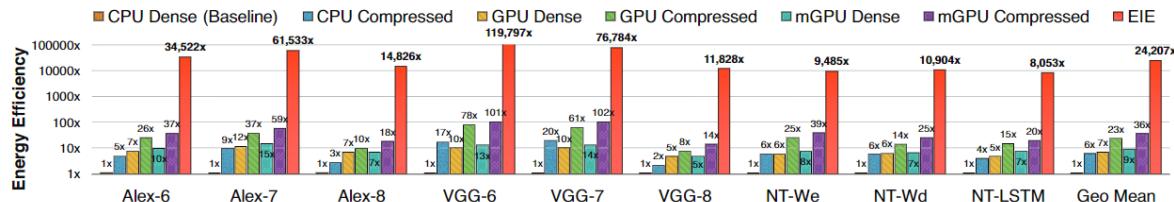
Segundo (THOMPSON; SPANUTH, 2018), em 2013 a empresa Google antecipou o aumento da demanda por voz e pesquisa por voz e projetou que o aprendizado profundo

Figura 3 – Aumento de velocidade de Processamento.



Fonte: (HAN et al., 2016).

Figura 4 – Aumento de eficiência energética (HAN et al., 2016).



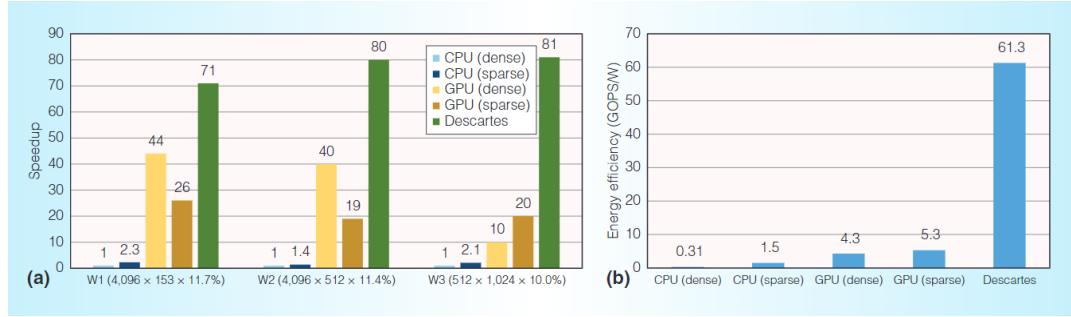
Fonte: (HAN et al., 2016).

exigiria uma duplicação da capacidade do seu data center. Em vez de adotar soluções convencionais, como processadores universais ou GPUs, o Google optou por desenvolver um processador altamente especializado chamado Tensor Processing Unit (TPU). Esses processadores personalizados são conhecidos como circuitos integrados específicos de aplicativos (ASICs). Embora a criação de um processador personalizado tenha sido um investimento considerável, estimado em dezenas de milhões de dólares, os benefícios foram significativos. O Google alega que o desempenho da TPU foi equivalente a sete anos de avanço previsto pela lei de Moore. Além disso, os custos evitados em infraestrutura compensaram o investimento inicial. Em 2017, o Google lançou a segunda geração da TPU, que era oito vezes mais rápida que as GPUs líderes de mercado, conforme medido pelo tempo necessário para treinar um modelo de tradução em larga escala.

Hardware de redes neurais que seja eficiente demanda métodos engenhosos para aproveitar os recursos disponíveis para alcançar alto poder de processamento ou baixo consumo (MISRA; SAHA, 2010). Pesquisas recentes com otimização (compressão do modelo) e projeto conjuntos de software e hardware mostram o desempenho superior de uma arquitetura dedicada chamada de ‘*Descartes*’ implementada na placa <sup>2</sup> Xilinx XC7Z02 comparada a CPU Core i7-5930k CPU e a GPU mobile Pascal TitanX GPU) (GUO et al., 2017).

<sup>2</sup>**FPGA (field-programmable gate array):** Circuito integrado projeto para ser reprogramável,

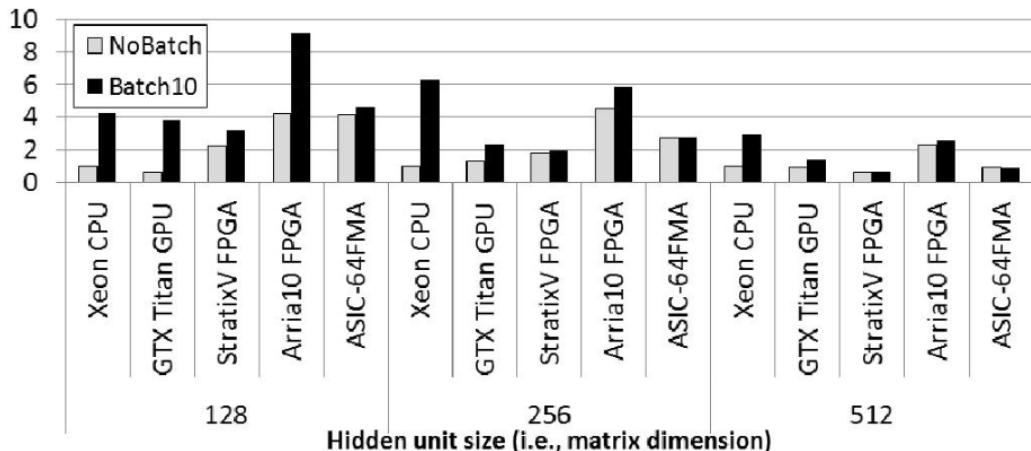
Figura 5 – Comparaçāo de desempenho entre arquitetura dedicada *Descartes* e CPU, GPU.



Fonte: (GUO et al., 2017).

Apesar de implementações dedicadas em ASIC ou FPGA apresentarem eficiência energética superior à CPU ou GPU, dependendo das implementações o desempenho por inferência pode ser inferior em uma FPGA dependendo do tamanho do *Batch*<sup>3</sup> e do modelo da placa FPGA (NURVITADHI et al., 2016).

Figura 6 – Comparaçāo de desempenho entre diversos aceleradores.



Fonte: (NURVITADHI et al., 2016).

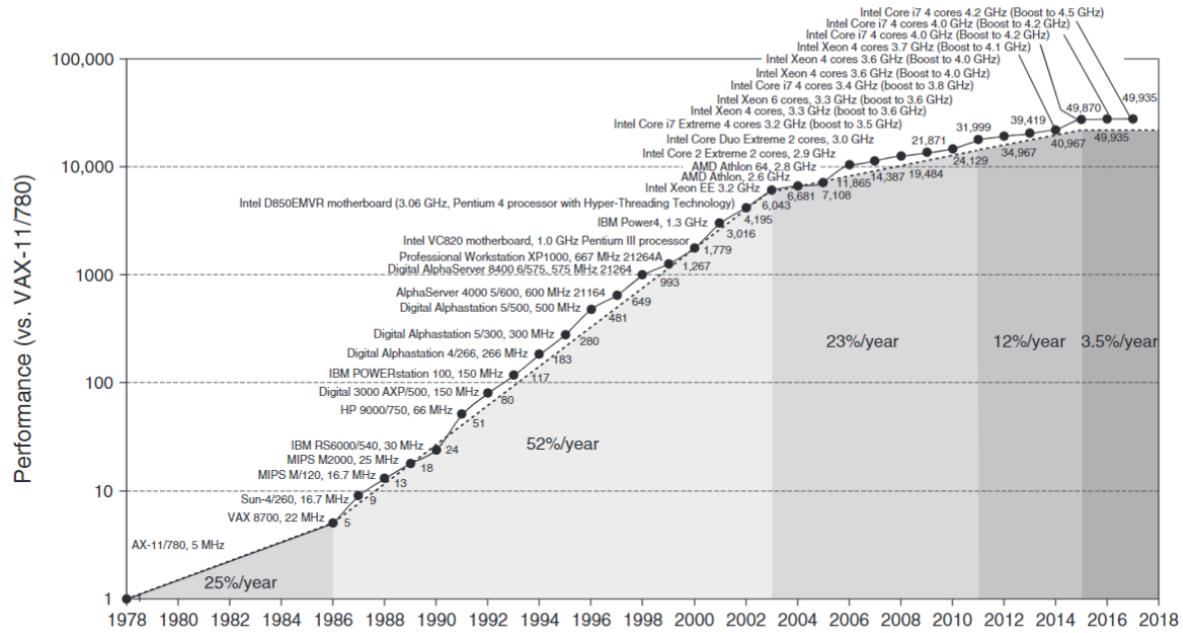
Os resultados acima citados, demonstram o poder de otimização no processamento de algoritmos de Inteligência Artificial com a construção de arquiteturas dedicadas para tal. Os avançados atraentes destas arquiteturas, somadas com a saturação da Lei de Moore

baseados em uma matriz de blocos reconfiguráveis (*configurable logic blocks - CLBs*).

<sup>3</sup>**Batch (lote):** O tamanho do lote é um hiper-parâmetro de gradiente descendente que controla o número de amostras de treinamento a serem trabalhadas antes que os parâmetros internos do modelo sejam atualizados.

(DENG et al., 2020), empurram o desenvolvimento de hardware para arquiteturas cada vez mais dedicadas, ou seja, diminuindo e vertente de arquiteturas genéricas de propósito geral (THOMPSON; SPANUTH, 2018).

Figura 7 – Lei de Moore e seu fim.



Fonte: (HENNESSY; PATTERSON; ASANOVIC, 2019).

## 1.2 Problema 2: Projeto de arquiteturas dedicadas

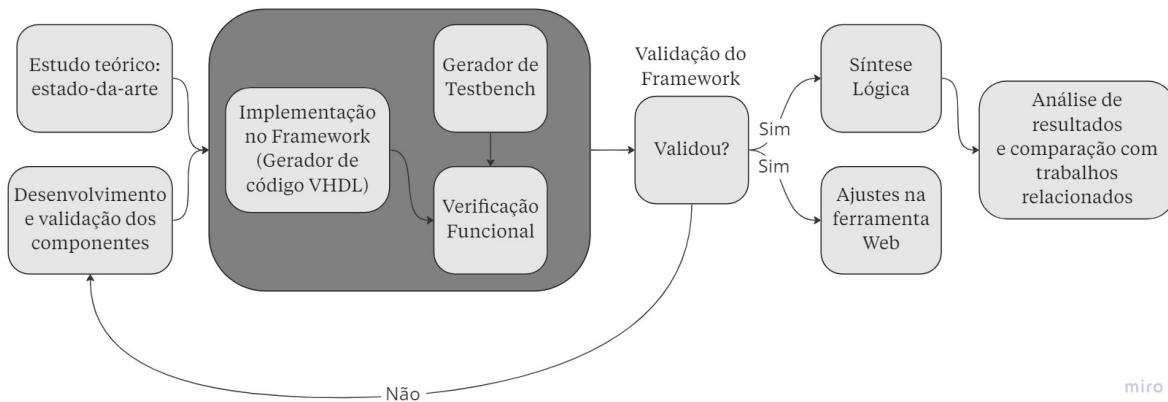
Embora a maioria das aplicações de Redes Neurais Artificiais (*Artificial Neural Networks* ANNs) existentes em uso comercial sejam frequentemente desenvolvidas apenas como software, existem aplicações específicas, como por exemplo detecção de fraude à bancos (conforme já citado na seção 1.1) e compressão de *streaming* de vídeo, que exige processamento em tempo real de alto volume e aprendizado de grandes conjuntos de dados em tempo razoável exigindo arquiteturas energeticamente eficientes com grande capacidade de processamento paralelo (LIU et al., 2022).

## 1.3 Metodologia

O desenvolvimento do Trabalho de Conclusão de Curso proposto seguirá as seguintes etapas:

- Estudo teórico: estado-da-arte;
- Desenvolvimento e Validação dos componentes
- Finalizar implementação genérica do *framework* (sem otimizações aritméticas);
- Estudar e implementar técnicas de baixo consumo de potência como e alto poder de processamento seguindo técnicas de Aritmética Digital com foco em arquiteturas de *AutoEncoders* de DNNs;
- Escolher melhores técnicas de Aritmética Digital e implementar estas opções no *framework* como opcionais ao usuário;
- Realização de simulações para verificação funcional de cada componente;
- Programar a Geração de *Testbench* e Verificação Funcional automatizados pelo *framework*;
- Implementar a síntese lógica e obter resultados estimados.

Figura 8 – Fluxo estimado de trabalho.



Fonte: Autor

Conforme mostrado na figura 8 simultaneamente a parte inicial se baseará no estudo teórico do estado-da-arte em trabalhos de mesmo foco e no desenvolvimento e validação de componentes de hardware implementados. Os estudo teórico terá foco em implementações de *Framework* para a síntese em alto-nível HLS de hardware dedicado

à algoritmos de Redes Neurais Artificiais e/ou síntese de baixo nível de arquiteturas dedicadas de mesmo propósito.

Após estas etapas, o próximo passo é o de implementação dos componentes e seus arquivos de teste no *Framework* em linguagem de programação *Python*, o qual tem a função de gerar descrição de hardware em linguagem VHDL. Haverá um processo iterativo, pois dependendo dos resultados de verificação funcional, na implementação de cada novo componente ou nova função, surgirá a necessidade ou não de novos ajustes. Caso tenha ocorrido tudo bem, será feita a síntese física utilizando bibliotecas Cadence.

Com os resultados de síntese, será feito a análise dos resultados e comparação com trabalhos relacionados. Estes resultados serão documentados ao final do trabalho.

## 1.4 Motivação e Objetivos

### 1.4.1 Motivação

A motivação envolve os problemas citados:

- GPUs são caras e não otimizadas para os algoritmos de IA, havendo espaço para otimização em eficiência energética e poder de processamento
- Para projetar arquiteturas dedicadas e otimizadas para cada tipo de arquitetura, é necessário conhecimento sobre os algoritmos de IA, conhecimentos em descrição de hardware (HDL) e tempo de projeto.

Conforme (COLUCCI et al., 2020), tendências da indústria e pesquisa avançaram na direção de IPs<sup>4</sup> e ASICs para aceleradores dedicados, tanto para inferência quanto para treinamento dos modelos de IA. O problema dos projetos baseados em ASIC é sua demora para chegar ao mercado, devido ao longo tempo de projeto e fabricação. Outro fator é que os algoritmos de IA estão em rápida evolução, trazendo a necessidade de novas extensões de hardware, o que é inviável devido ao curto período de tempo.

Desta forma um *framework* para geração de código VHDL otimizado para o modelo treinado, traz uma arquitetura otimizada sem a necessidade de conhecimentos em descrição de hardware, em muito menor tempo de projeto, ainda trazendo opções de

---

<sup>4</sup>**IP:** Núcleo de propriedade intelectual (núcleo IP) é um bloco funcional de lógica ou dados usados para criar um FPGA ou um circuito integrado de aplicação específica para um produto (ASIC). Comumente usado em semicondutores, um núcleo IP é uma unidade reutilizável de lógica ou projeto de layout de circuito integrado (IC) (AWATI, 2022).

customizações (para os mais chegados em hardware digital). Resumidamente, a proposta surge para diminuir a complexidade de projeto de uma arquitetura dedicada, com a implantação e otimização automática de aceleradores de ANNs.

### 1.4.2 Objetivos Gerais

O objetivo deste trabalho é desenvolver um *framework* para geração automatizada de código VHDL sintetizável para algoritmos de Redes Neurais Artificiais e estudar o efeito de diferentes configurações de arquiteturas no desempenho do algoritmo, assim como seu consumo de área e potência, desta forma, trazendo um ‘guia’ para o usuário na hora de ajuste dos parâmetros no uso do *framework*, dependendo de suas preferências e modelo.

### 1.4.3 Objetivos Específicos

O presente estudo tem como objetivo principal o desenvolvimento de um framework para construção de ferramentas no contexto de arquiteturas de redes Neurais Artificiais de modelo MLP (*Multi-Layer Perceptrons*), que envolvem arquiteturas de multiplicação e acumulação, bem como de funções de ativação.

Outro objetivo é permitir ao usuário a implementação de seus próprios multiplicadores e somadores aritméticos, criados por ele mesmo.

Além da construção das ferramentas no framework, pretende-se realizar um estudo das diferentes arquiteturas geradas. Será realizada uma análise comparativa dessas arquiteturas, levando em consideração o consumo energético, a ocupação de área e a latência. O objetivo é encontrar revelações que possam orientar os usuários do framework na escolha da arquitetura mais adequada de acordo com a aplicação desejada.

## 2 Referencial Teórico

Este capítulo apresenta uma visão geral sobre os conceitos de Machine Learning (ML) e Deep Learning, destacando suas aplicações e características principais. Serão mencionadas as técnicas de treinamento em ML, como o aprendizado supervisionado, não-supervisionado e por reforço.

Em seguida, é destacada a estrutura das ANNs, com suas camadas de neurônios interconectados, e a importância das funções de ativação na obtenção de respostas não-lineares.

Este capítulo fornecerá uma base teórica necessária para compreender os temas abordados nos próximos capítulos, que se concentra em ferramentas e arquiteturas relacionadas ao desenvolvimento de modelos de Deep Learning.

### 2.1 *Machine Learning*

*Machine Learning* (ML) é uma disciplina da área da Inteligência Artificial que, por meio de algoritmos, dá aos computadores a capacidade de identificar padrões em dados e fazer previsões (análise preditiva). Essa aprendizagem permite que os computadores efetuam tarefas específicas de forma autônoma, ou seja, sem necessidade de serem programados, apenas treinados, com ou sem supervisão (IBERDROLA.COM, 2022).

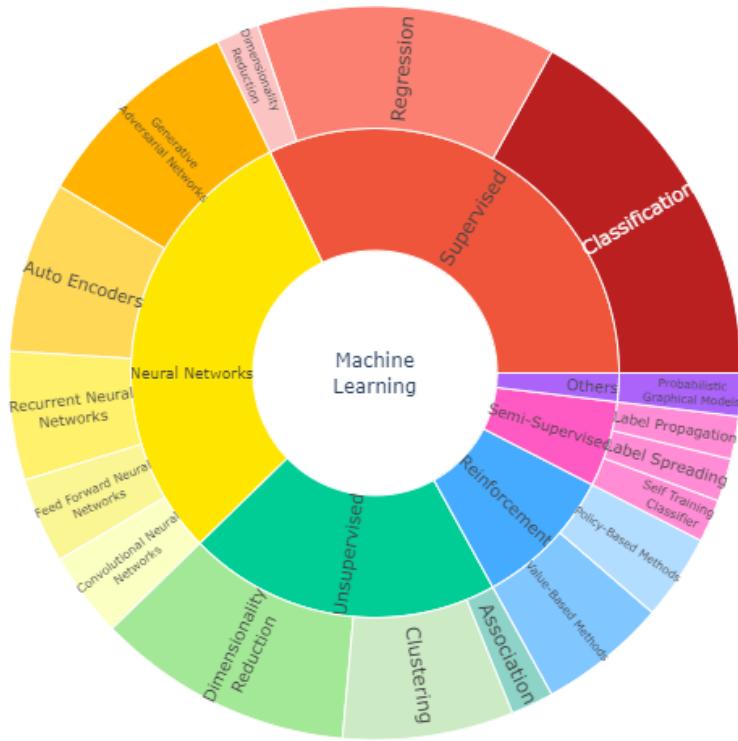
Modelos de aprendizado de máquina (*Machine Learning*) se diferenciam por suas estruturas e aplicações.

ML tradicionalmente possui 3 técnicas de treinamento:

1. Aprendizado supervisionado
2. Aprendizado não-supervisionado
3. Aprendizado por reforço

O que diferencia cada uma é que a primeira contém os rótulos das respostas corretas, a segunda não, ou seja, quando não há um gabarito de respostas corretas. Já no aprendizado por reforço, o aprendizado acontece através da interação com um ambiente, que fornece punições e recompensas, o qual a última o algoritmo tenta maximizar.

Figura 9 – Classificação de algoritmos de aprendizado de máquina.



Fonte: (DOBILAS, 2022).

## 2.2 Deep Learning

Deep Learning é uma categoria da disciplina de ML que faz o uso de Redes Neurais Artificiais (Artificial Neural Networks ANNs).

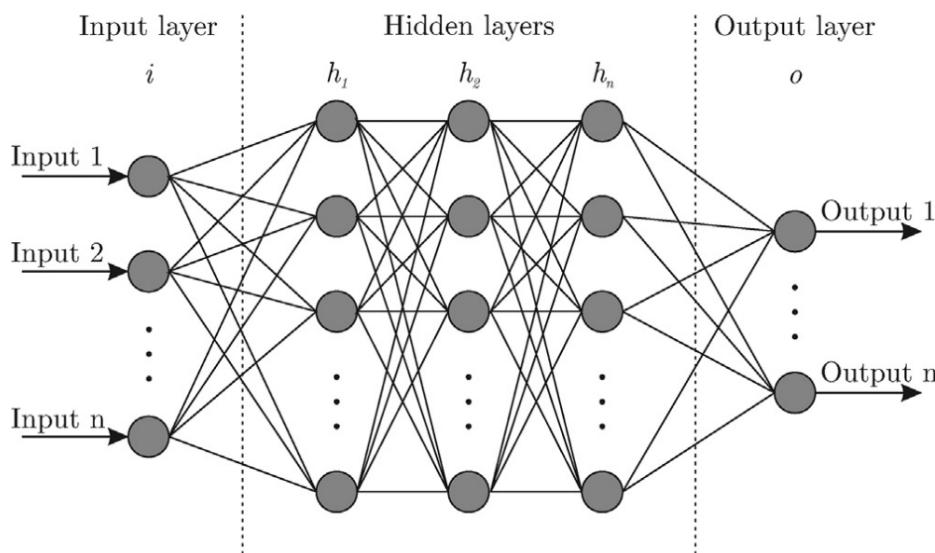
ANNs são nada mais do que uma rede de nodos interligados chamados de neurônios artificiais. Em cada camada da ANN os neurônios recebem múltiplas entradas diferentes vindas da camada anterior da ANN e saídas iguais para os neurônios da próxima camada.

Na Figura 10 podemos ver a estrutura genérica de uma ANN, constituída pela camada de entrada (*Input Layer*), camadas intermediárias ou ocultas (*Hidden layers*) e a camada de saída (*Output layer*).

O que diferencia uma ANN de outra são diversos fatores, dentre os mais importantes: o número de camadas, número de neurônios em cada camada, valores dos pesos e viés de cada neurônio, funções de ativação de cada camada.

ANNs com um número muito grande de camadas ocultas são conhecidas como DNNs (*Dense Neural Networks*) ou em outras vezes MLPs (*Multi-Layer Perceptrons*).

Figura 10 – Arquitetura de uma Rede Neural Artificial genérica (ANN  $i \rightarrow h_1 \rightarrow h_2 \rightarrow h_n \rightarrow o$  ).



Fonte: (BRE; GIMENEZ; FACHINOTTI, 2018).

### 2.2.1 Perceptron

Os neurônios possuem uma estrutura interna que se diferencia do seu modelo. Neurônios artificiais conhecidos como Perceptron, multiplicam cada uma das diferentes entradas por diferentes pesos e somam os resultados de todas as multiplicações com um valor de deslocamento conhecido como *bias*. Este processo de multiplicações e somas é conhecido como MAC (*Multiply And Accumulate* ).

$$y = f \left( \sum_{j=1}^n x_j \cdot w_j + \gamma \right) \quad (2.1)$$

O modelo matemático do funcionamento de um Perceptron é apresentado na equação 2.1. O diagrama de blocos de um Perceptron é representado na Figura 11, onde os pesos são indicados como  $w_1, w_2, \dots, w_j$ , os dados de entrada são referidos como  $x_1, x_2, \dots, x_j$ , o viés é representado por  $\gamma$  e a função de ativação é  $f$ .

### 2.2.2 Funções de Ativação

Após todo este processo, o valor resultante do MAC passa por uma função de ativação, que permite saídas não-lineares às entradas. O poder de aprendizado e generalização dos dados de treinamento do algoritmo de *Deep Learning* é graças à resposta não-linear

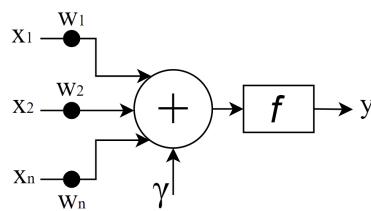
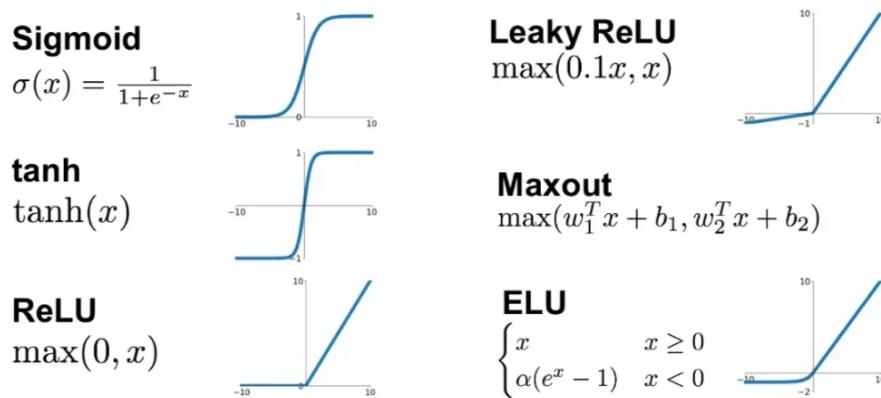


Figura 11 – Diagrama de blocos do Perceptron.

das funções de ativação de cada neurônio. Esta diferença dos demais algoritmos é uma das principais características que diferencia as Redes Neurais Artificiais de outros algoritmos de ML em seu potencial e aplicações.

Figura 12 – Exemplo de Funções de Ativação.



Fonte: (JADON, 2018).

Funções de ativação definem a saída de um nó dada uma entrada ou conjunto de entradas. Funções de ativação lineares nos fornecem Perceptrons lineares. Funções de ativação não lineares, tornam os Perceptrons não-lineares e permitem que essas redes calculem problemas não triviais usando apenas um pequeno número de nós (HINKELMANN, 2022).

Dentre as funções de ativação possíveis, três das mais conhecidas e comumente usadas são a ReLU (*Rectified and Logic Unit*), Sigmoide ou Logistica e Leaky ReLU, porém muitos outros tipos existem.

### 2.2.3 Autoencoders

Modelos conhecidos como *AutoEncoders*, são utilizados principalmente para aprendizado não-supervisionado, ou seja, quando não há um ‘gabarito’ de respostas corretas.

Desta forma o algoritmo encontra os padrões existentes nos dados e suas correlações.

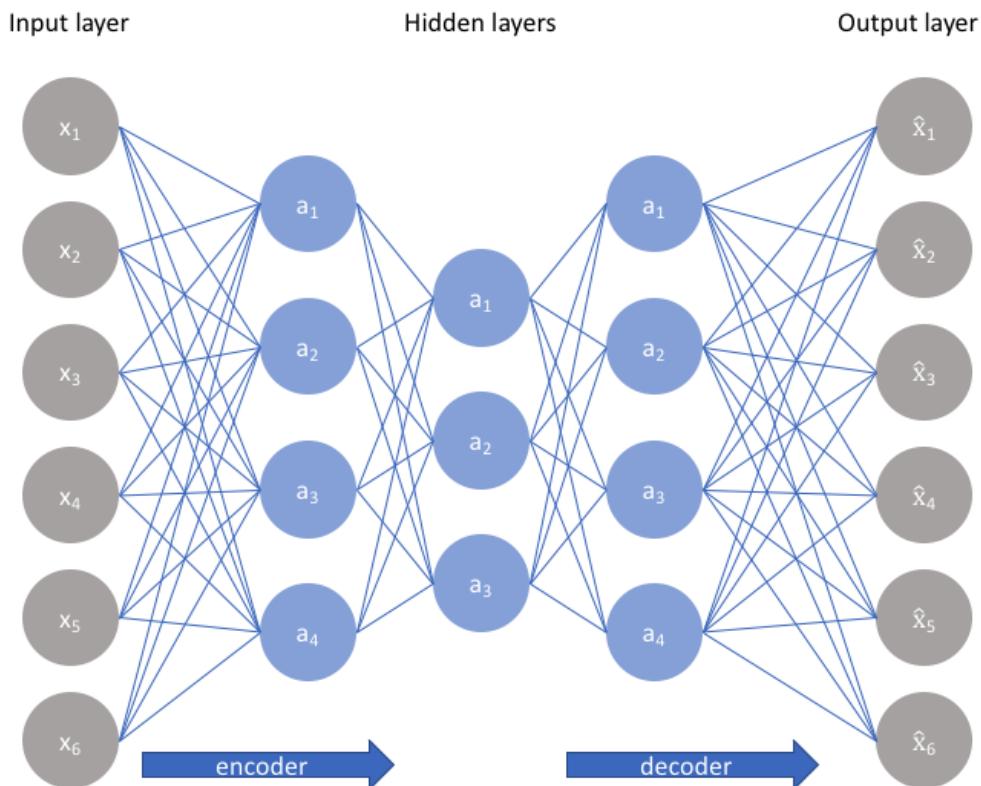
Segundo Academy os *Autoencoders* (AE) são redes neurais que visam copiar suas entradas para suas saídas. Eles trabalham compactando a entrada em uma representação de espaço latente e, em seguida, reconstruindo a saída dessa representação.

Esse tipo de rede é composto de duas partes:

Codificador (*Encoder*): é a parte da rede que compacta a entrada em uma representação de espaço latente (codificando a entrada). Pode ser representado por uma função de codificação  $h = f(x)$ .

Decodificador (*Decoder*): Esta parte tem como objetivo reconstruir a entrada da representação do espaço latente. Pode ser representado por uma função de decodificação  $r = g(h)$ , Academy (2022).

Figura 13 – Arquitetura genérica de um *AutoEncoder*.



Fonte: (JORDAN, 2018).

*AutoEncoders* são utilizados para compressão e/ou descompressão de informação, possuindo diversas aplicações, como por exemplo em codificadores de vídeo (HABIBIAN et al., 2019), redução de dimensionalidade (AMORIM, 2022), detecção de anomalias (NASCIMENTO et al., 2021) e remoção de ruídos (DSA, 2019).

### 3 Trabalhos relacionados

Conforme Lebedev e Belecky (2021), atualmente existem diversas ferramentas e frameworks para algoritmos de inteligência artificial as quais podem ser divididas em 3 grupos: Ferramentas que otimizam e preparam um modelo de Rede Neural Artificial:

1. Para ser sintetizável em um modelo RTL<sup>1</sup> que pode tanto ser executado em uma FPGA quanto implementado em <sup>2</sup> (sintetizadores em RTL);
2. Em um executável binário para um processador de arquitetura específica (compiladores);
3. Para placas de propósito específico como FPGAs (sintetizadores para placas específicas);

Assim como a proposta deste presente trabalho, a revisão do estado da arte engloba apenas trabalhos do terceiro grupo, ou seja, modelos sintetizáveis em RTL.

#### 3.0.1 Vitis AI

Vitis AI é um *framework* desenvolvido pela Xilinx que na verdade pertence aos grupos 1 e 2, ou seja, possui compiladores e sintetizadores para placas específicas da Xilinx, como Alveo, Zynq UltraScale+, and partly Zynq-7000 (LEBEDEV; BELECKY, 2021). Seu sistema também possui otimizadores e compressores para os modelos aceitos de Redes Neurais Artificiais como, Caffe, Pytorch e Tensorflow. Este *framework* possui código aberto, porém nem todas as suas soluções estão disponíveis, pois os núcleos de DPUs<sup>3</sup> são criptografados ou estão disponíveis apenas com a devida licença. É um *framework* disponível para uso comum e possui tutoriais disponíveis ao usuário (XILINX, 2022a).

Na Figura 14 pode ser visto a arquitetura DPUCZDX8G, utilizando a placas da categoria Zynq UltraScale+ MPSoC. Apesar deste *framework* não pertencer ao grupo 3, foco deste trabalho conforme comentado anteriormente, possíveis comparações de

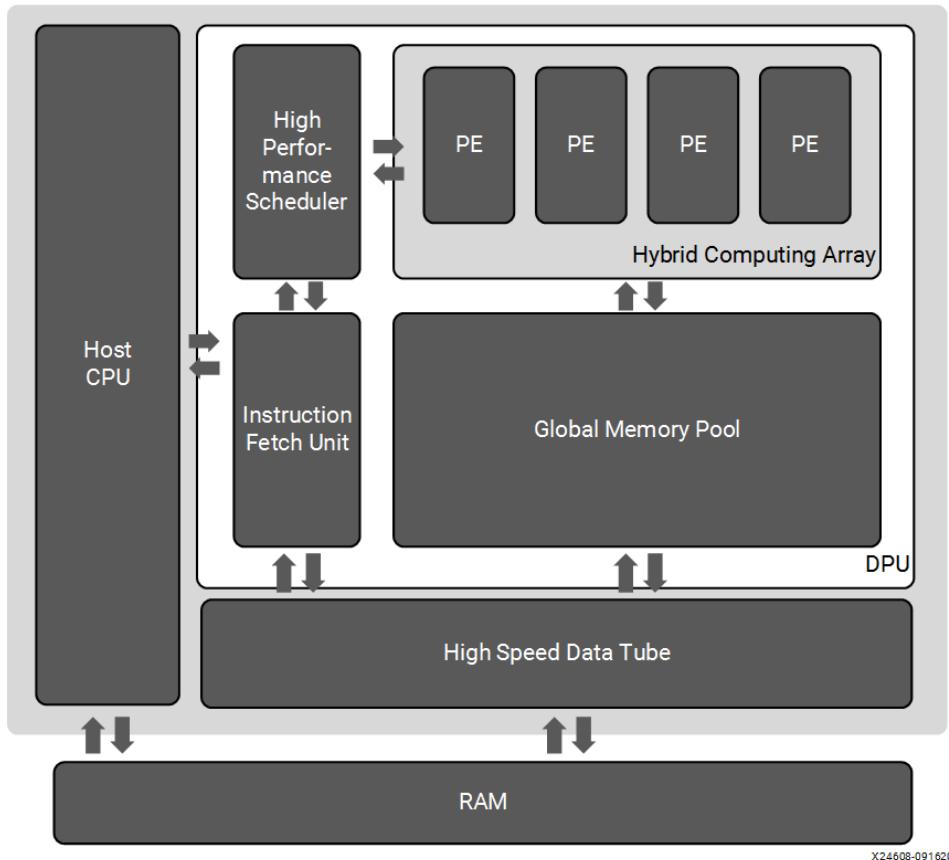
---

<sup>1</sup> **Register Transfer Level (RTL):** RTL é uma abstração dos componentes digitais de um circuito de hardware. É a principal abstração usada para definir sistemas eletrônicos atualmente e geralmente serve como modelo de ouro no fluxo de design e verificação (SEMIENGINEERING.COM, 2021).

<sup>2</sup> **ASICs:** Circuitos integrados de aplicação específica (*application-specific integrated circuit ASIC*) é um chip de circuito integrado projetado para uma finalidade específica.

<sup>3</sup> **DPU (Deep Learning Processing Unit):** DPUs são componentes digitais de processamento de algoritmos de Deep Learning (Aprendizado profundo).

Figura 14 – Arquitetura DPUCZDX8G para placas da categoria Zynq UltraScale+



Fonte: (XILINX, 2022b).

resultados poderão ser feitas futuramente, uma vez que o Laboratório de Computação Embarcada ECL (Embedded Computing Laboratory) na UFSC possui uma unidade da placa Zynq-7000 disponível para uso.

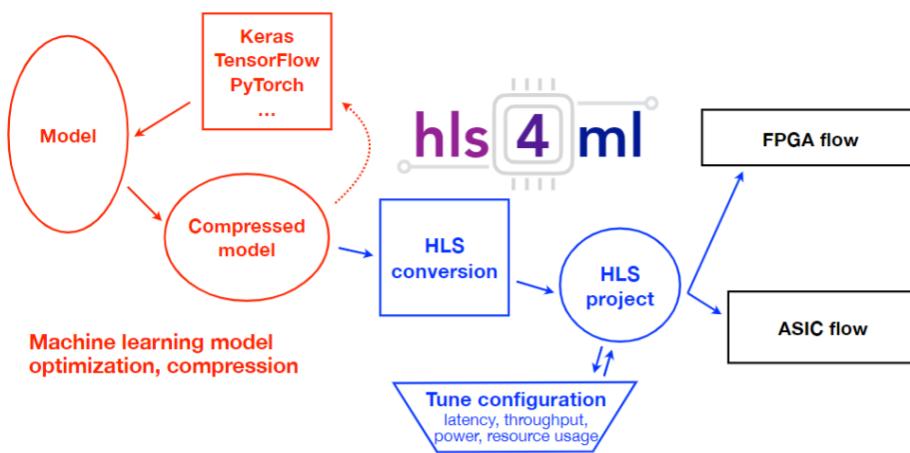
### 3.1 Ferramentas de código-aberto

#### 3.1.1 hls4ml

hls4ml é um *framework* de código aberto (*open-source*) sendo um dos mais completos. Seu sistema aceita modelos de Rede Neural Artificial em diversos formatos de entrada, como ONNX, Tensorflow, Keras e Pytorch, sendo as bibliotecas mais conhecidas e utilizadas.

zadas atualmente, otimiza estes modelos, como *quantization-aware training*<sup>4</sup> e *Pruning*<sup>5</sup>, implementa a conversão em síntese de alto-nível (HLS<sup>6</sup> em linguagem C/C++, implementa ajuste finos dependendo do foco de arquitetura e de projeto, analisando latência, desempenho e consumo de potência, consumo de recursos (componentes disponíveis na placa desejada ou área ocupada em uma síntese ASIC) trazendo ainda a possibilidade de implementações com foco em baixo consumo energético. O fluxo de trabalho do framework hls4ml pode ser visto na Figura 15.

Figura 15 – Fluxo de trabalho do framework hls4ml.



Fonte: (FAHIM et al., 2021).

### 3.1.2 NNGen

NNGen é um *framework* de código aberto que aceita modelos descritos em formato ONNX ou utilizando uma descrição própria em linguagem de programação Python. O

<sup>4</sup>**Pruning (Poda):** Esta técnica consiste em retirar partes do modelo que menos colaboram para o resultado de saída do modelo, com foco em reduzir o número de área consumida e operações realizadas como por exemplo: neurônios "mortos" em que sua saída está sempre tendendo à valores muito baixos (ocorre muito com a função de ativação ReLU) que resultariam em pouco impacto na camada posterior. A técnica de Poda também pode ser realizada para retirar partes do modelo com valores muito altos, buscando assim a diminuição do consumo energético (SZE et al., 2017).

<sup>5</sup>**Quantization-aware training:** Esta é uma treinamento do modelo já consciente da quantização dos pesos da rede. Estudos mostram que quantizar os pesos durante o treinamento trazem uma acurácia melhor comparada à treinar o modelo normalmente e só ao final quantizar os pesos do modelo (FAHIM et al., 2021).

<sup>6</sup>**HLS (High Level Synthesis) :** A síntese de alto nível (HLS), às vezes chamada de síntese algorítmica ou síntese comportamental, é um processo de design automatizado que pega uma especificação comportamental abstrata de um sistema digital e encontra um registro -estrutura de nível de transferência (RTL) que realiza o comportamento dado Wikipedia (2022a).

hardware gerado é completo, o que inclui mecanismo de processamento, memória on-chip, rede on-chip, controlador DMA<sup>7</sup> e circuitos de controle. Portanto, o hardware gerado não requer nenhum controle adicional de um circuito externo ou da CPU após o início do processamento (YAMAZAKI, 2022).

## 3.2 Tabela Comparativa dos principais *Frameworks*

Uma análise qualitativa de trabalhos similares foi feita e pode ser vista na tabela 8 no apêndice A. A coluna 'selecionados', indica se o trabalho foi ou não comentado no capítulo 3. Importante denotar que o único trabalho que realiza a síntese em ASIC automaticamente é o *framework hls4ml*, o qual possui sua última atualização no ano de 2021 porém demonstra ser a ferramenta com mais funcionalidades (o que não necessariamente traz as melhores arquiteturas).

COLOCAR TABELA MENOR DOS PRINCIPAIS!!!!!!

---

<sup>7</sup>DMA: O DMA ou *Direct Memory Access* é o método que permite que um dispositivo de entrada e saída envie ou receba dados diretamente da memória principal, ignorando a CPU, acelerando as operações que envolvem a memória (TECNOBLOG, 2022).

# 4 Proposta

## 4.1 Framework

A proposta se baseia na construção de um *framework* HLS<sup>6</sup> para a rápida síntese e prototipação de arquiteturas de algoritmos de Redes Neurais Artificiais de estrutura MLP<sup>1</sup>, utilizando linguagem de programação Python e descrição de hardware VHDL. O mesmo permite a exploração de diferentes configurações de arquiteturas através de configurações parametrizáveis e modelos de entrada e a possibilidade ao usuário da implementação de novas estruturas (código-aberto). As interfaces entre usuário e o sistema proposto deverá se apresentar em 2 formas:

- **Web-framework:** versão mais acessível a usuários sem conhecimento de linguagens HDL e Python
- **Código-aberto:** versão que disponibiliza novas implementações e modificações a usuários mais imersos em descrição de hardware e linguagem Python.

## 4.2 Melhorias para Arquiteturas Geradas pelo Framework

Este capítulo aborda a investigação de possíveis melhorias para as arquiteturas geradas pelo framework desenvolvido neste trabalho. Devido a restrições de tempo para a entrega do presente trabalho de conclusão de curso (TCC), não foi possível realizar testes abrangentes para verificar se as arquiteturas geradas são 100% funcionais, necessitando talvez de pequenas correções para manter a precisão dos modelos. No entanto, essa limitação oferece uma oportunidade valiosa para explorar potenciais áreas de aprimoramento e sugestões para trabalhos futuros, além de fornecer contribuições para outros pesquisadores interessados na criação de arquiteturas de Redes Neurais Artificiais (RNAs) que fazem uso de unidades aritméticas de multiplicação e soma, conhecidas como MACs (Multiply and Accumulate) para redes multi-camadas de perceptrons (MLPs).

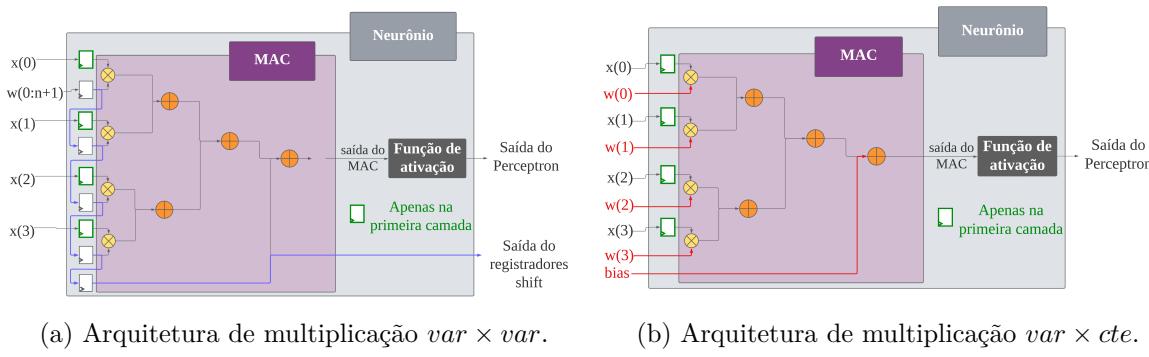
---

<sup>1</sup>**MLP - Multi-Layer Perceptron :** Um multilayer perceptron (MLP) é uma classe totalmente conectada de rede neural artificial (ANN) *feedforward* (Multilayer perceptron, 2022), ou seja, suas conexões não formam um ciclo.

Como as arquiteturas geradas pelo framework, possuem sua unidades aritméticas de multiplicações e somas (MACs) combinacionais, esta abordagem traz um grande consumo de área, pois traz um multiplicador individual para cada operação de multiplicação e um somador para cada operação de adição.

Logo baseado neste grande consumo de área, a investigação para aprimorar as arquiteturas geradas pelo framework concentrou-se em duas abordagens principais: a substituição dos multiplicadores dos MACs por blocos de somas e deslocamentos (*shifts*) e a troca de todos os somadores por um bloco de compressão dos vetores. Essas melhorias têm como objetivo otimizar o desempenho e a eficiência das arquiteturas geradas, levando em consideração as restrições de recursos e as necessidades específicas dos sistemas embarcados que utilizam essas RNAs.

Figura 16 – Arquiteturas de multiplicação combinacional (pesos variáveis vs pesos fixos).



Fonte: Autor.

Entretanto, é importante destacar que a comparação não pode ser considerada completamente justa. Na primeira versão da rede neural, que é a única gerada pelo framework, os valores de entrada e pesos são variáveis na arquitetura, devido os pesos e viés poderem ser alterados a qualquer momento, pois estão salvos nos registradores, que podem ser atualizados sempre que necessário, conforme Figura 16a. Já na segunda versão, apenas as entradas eram variáveis, com os pesos sendo agora fixos pois estão contidos nas arquiteturas fixas de somas e deslocamentos (destacados em vermelho). Logo para uma comparação justa, as análises posteriores fazem a comparação com a arquitetura de multiplicação de variáveis por constantes, entradas e pesos respectivamente, conforme Figura 16b.

### 4.2.1 Substituição dos Multiplicadores por Blocos de Somas e Deslocamentos

Uma das estratégias investigadas consiste na substituição dos multiplicadores dos MACs por blocos de somas e deslocamentos. Essa abordagem se baseia na observação de que, em algumas aplicações de RNAs, certos padrões de multiplicação podem ser representados de forma mais eficiente por operações de soma e deslocamento. A substituição dos multiplicadores por blocos de somas e deslocamentos pode reduzir a complexidade dos circuitos gerados, diminuindo a área ocupada e o consumo de energia. As arquiteturas foram geradas com o auxílio da ferramenta Spiral, de autoria de Yevgen Voronenko para o projeto Spiral, pela universidade de Carnegie Mellon. Uma das abordagens eficientes para a multiplicação de uma variável por um conjunto de constantes em ponto fixo é por meio de um bloco multiplicador que consiste exclusivamente de adições, subtrações e deslocamentos. A geração desse bloco multiplicador a partir do conjunto de constantes é conhecida como o problema da multiplicação por múltiplas constantes (*Multiple Constant Multiplication - MCM*). Encontrar a solução ótima, ou seja, aquela que requer o menor número de adições e subtrações, é conhecido por ser um problema NP-completo.

Um exemplo de equação que representa a equivalência de uma multiplicação por somas e deslocamento é:

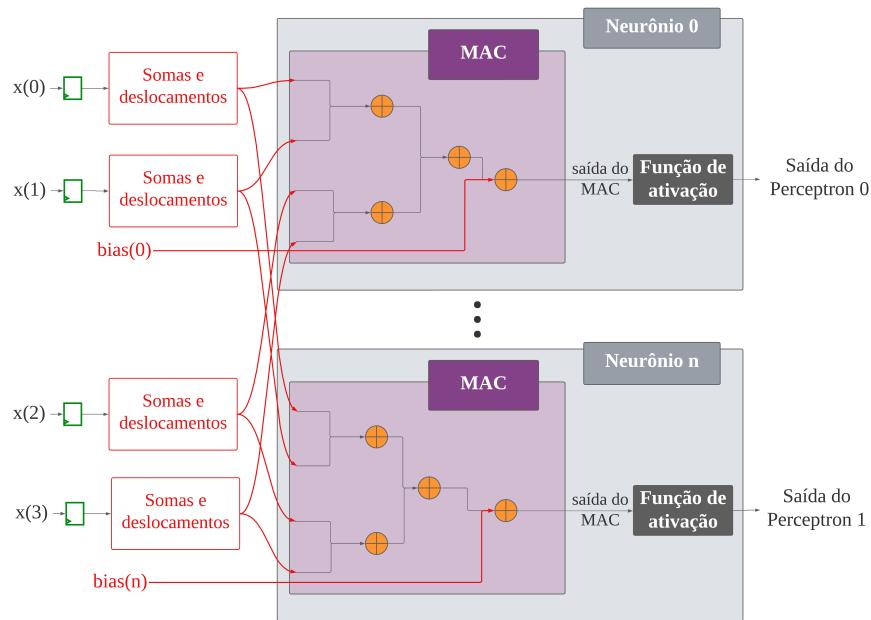
$$9a = 8a + a \quad (4.1)$$

Nesta equação,  $a$  é a entrada variável que pode assumir qualquer valor e 9 é a constante de multiplicação. Nas arquiteturas de RNNs estas constantes são os pesos e viés (bias) dos neurônios.

A Figura 17 apresentada ilustra a adaptação da arquitetura de múltiplos neurônios em uma rede neural, onde é aproveitada a vantagem de compartilhar as mesmas entradas da camada anterior para cada neurônio, enquanto cada neurônio possui seus próprios pesos e viés individuais. Para implementar todas as multiplicações correspondentes às mesmas entradas, provenientes de neurônios diferentes, a arquitetura utiliza blocos de somas e deslocamentos em vez de multiplicadores.

Na Figura 17 mostra por exemplo a utilização desses blocos de somas e deslocamentos para a entrada  $x(0)$ , onde um único bloco é responsável por realizar todas as operações de multiplicação necessárias para cada neurônio da camada. Cada saída do bloco de somas e deslocamentos é conectada a um neurônio específico na camada, permitindo que os pesos individuais de cada neurônio sejam aplicados às entradas correspondentes.

Figura 17 – Blocos de soma e deslocamentos (Spiral).



Fonte: Autor.

Essa abordagem proporciona eficiência na implementação, pois reduz a necessidade de multiplicadores individuais para cada neurônio e aproveita o compartilhamento de entradas. Ao agrupar as operações de multiplicação em um único bloco de somas e deslocamentos relacionado a cada entrada, a arquitetura simplifica o circuito e melhora a eficiência computacional.

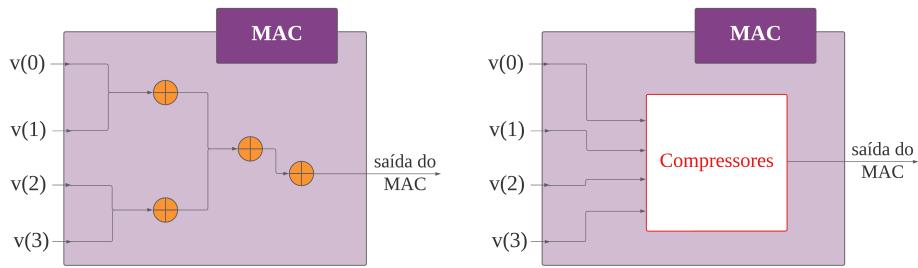
Essa figura representa uma solução alternativa e eficaz para implementar a multiplicação em arquiteturas de redes neurais, permitindo o processamento paralelo e otimizado de múltiplos neurônios em uma única camada.

#### 4.2.2 Troca dos Somadores por um Bloco de Compressão de Vetores

Esta abordagem visou a substituição de todos os somadores presentes nas arquiteturas geradas por um bloco de compressão de vetores, conforme exemplificado na Figura 18. Na arquitetura gerada pelo framework valores multiplicados são acumulados com uma árvore somadora, que possui níveis de somadores  $\log_2(N)$ . Em seguida, é realizada uma adição final com o viés, após disso a saída MAC está pronta.

Essa estratégia se baseia no conceito de que, em muitas aplicações de RNAs, os soma-

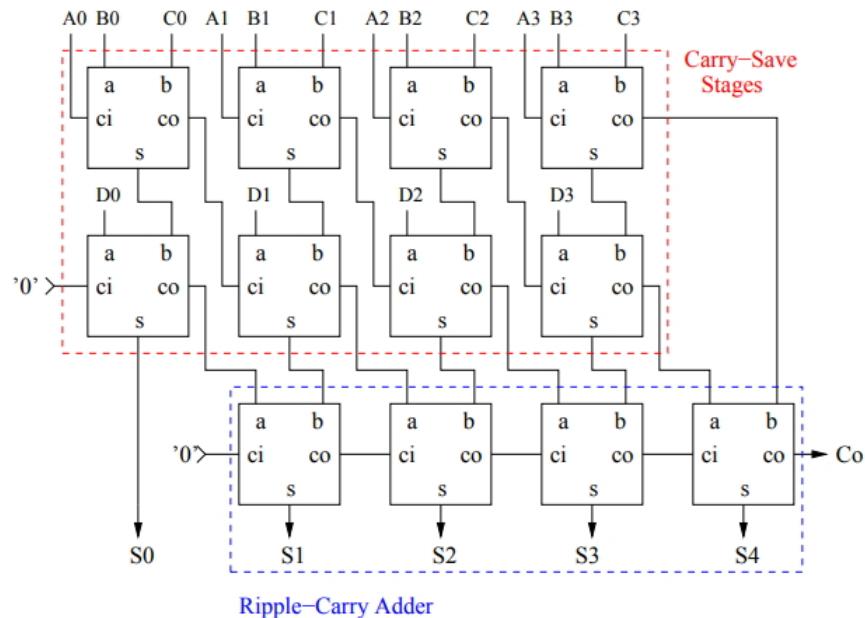
Figura 18 – Substituição das somas por um bloco de compressores.



Fonte: Autor.

dores convencionais podem ser substituídos por técnicas mais eficientes de compressão de dados. O objetivo dessa substituição é reduzir a área ocupada pelos somadores a troca de um provável maior consumo de energia e tempo de processamento.

A investigação envolveu o uso de compressores conhecidos como CSAs (*Carry-Save Adders*), arquitetura que possui uma série de compressores em cascata dos múltiplos vetores de soma da operação do MAC, para ao final, restando apenas 2 vetores, os mesmos são somados com um somador *Ripple-Carry*, conforme Figura 19.

Figura 19 – *Carry-Save Adders*.

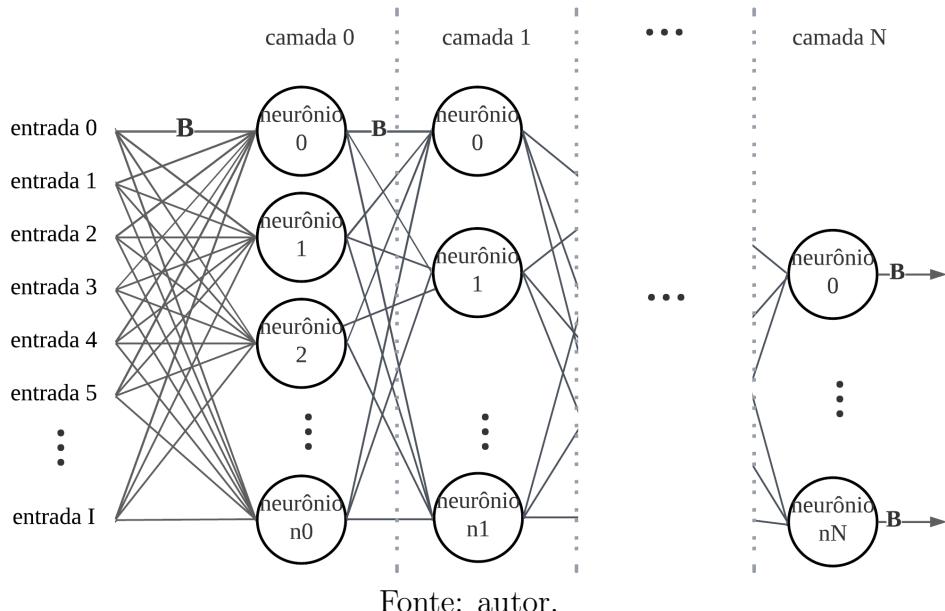
Fonte: (GYAN, 2022).

# 5 Resultados

## 5.1 Framework

Os resultados alcançados durante este projeto, trazem um framework que permite a geração de código VHDL para redes MLP com qualquer número arbitrário  $N$  de camadas,  $I$  entradas e  $n$  neurônios, conforme Figura 20. Essa flexibilidade oferece uma vantagem significativa aos projetistas, permitindo a criação de redes neurais com arquiteturas personalizadas para atender às necessidades específicas de cada projeto. Além disso, essa capacidade de geração automatizada de código VHDL reduz consideravelmente o tempo necessário para o desenvolvimento de redes neurais extensas.

Figura 20 – Arquitetura genérica de uma Rede Neural Artificial de Perceptrons multi-camadas (MLPs) gerada pelo framework.



### 5.1.1 Personalização da Largura de Representação de Bits

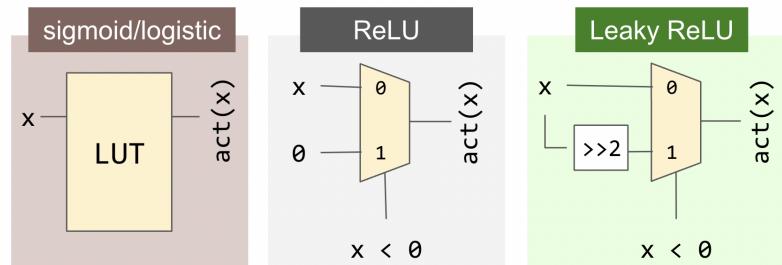
Outra funcionalidade importante do framework é a possibilidade de personalização da largura de representação de bits  $B$  nas unidades aritméticas. Isso permite que o projetista ajuste a precisão numérica de acordo com os requisitos do projeto, encon-

trando um equilíbrio entre desempenho e consumo de recursos. COLOCAR GRÁFICO COMPARAÇÃO LARGURA DE BITS SIM/ISCAS

### 5.1.2 Escolha de Funções de Ativação

O framework oferece três opções de funções de ativação: ReLU, Leaky ReLU e Sigmoid como LUT (Look Up Table), conforme Figura 12. Essa escolha permite que o projetista selecione a função de ativação mais adequada para cada camada da rede neural, considerando as características do problema em questão. Essa flexibilidade adiciona uma camada de personalização às redes neurais geradas, ampliando suas capacidades de representação.

Figura 21 – Funções de ativação disponíveis



Fonte: autor.

### 5.1.3 Inclusão de Barreiras de Registradores

O framework também permite a inclusão de barreiras de registradores nas unidades aritméticas dos neurônios, que podem ser vistos como registradores em azul, na Figura 24. Essa funcionalidade visa otimizar o atraso crítico da rede neural, reduzindo o tempo necessário para propagar os sinais através das unidades aritméticas. No entanto, é importante ressaltar que essa otimização resulta em um aumento no consumo de área e potência da rede neural.

COLOCAR GRÁFICOS DE COMPARAÇÃO DE GANHO ENTRE COLOCAR BARREIRAS E NÃO COLOCAR

PODERIA HAVER A SÍNTESE DE DIVERSAS CONFIGURAÇÕES PARA TRAZER OS RESULTADOS

### 5.1.4 Modos de Atualização de Pesos e Inferência

As arquiteturas de redes neurais implementadas pelo framework proposto possuem dois principais modos de operação: (i) atualização de pesos, onde os pesos e vieses armazenados em cada camada são atualizados com novos valores, e (ii) inferência, quando a rede realiza a propagação dos dados de entrada através da rede, produzindo as saídas previstas. A motivação para isso decorre do fato de que os modelos de redes neurais estão constantemente sendo atualizados, sendo importante fornecer mecanismos que permitam aos usuários modificar esses valores sempre que necessário.

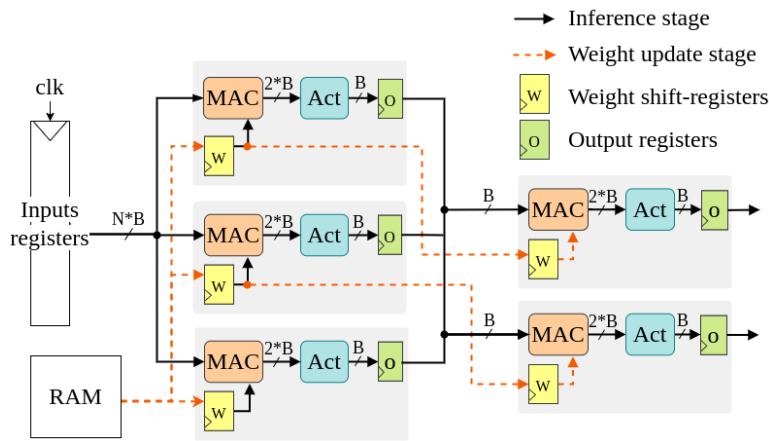


Figura 22 – Exemplo de uma arquitetura em hardware de um MLP gerado pelo framework proposto.

Um exemplo de arquitetura de hardware geral com uma camada oculta com 3 neurônios ( $n = 3$ ) e uma camada de saída com 2 neurônios ( $O = 2$ ) é mostrado na Figura 22. O retângulo azul representa a unidade de ativação. Para reduzir a congestão de roteamento, a arquitetura desloca os dados pelos registros de peso durante a fase de atualização de pesos. A estrutura dos registradores de deslocamento (*shift-registers*) é ilustrada na Figura 23, onde a seção amarela, identifica o bloco registrador 'w' da Figura 22, ou seja, o bloco 'w' na verdade é um composto de registradores individuais (um para cada peso). As conexões laranjas, são utilizadas apenas na etapa de atualização dos pesos, que possuem conexão entre si, dessa forma cada neurônio só irá ter uma entrada de  $B$  bits de largura. Esse deslocamento de pesos entre os registradores, seguirá até a última camada, conforme ilustrado na Figura 22.

Portanto, para  $L$  camadas ocultas, a atualização de pesos irá requerer  $(N + 1) + \sum_{i=2}^{L+1} (n_{i-1} + 1)$  ciclos para ser concluída, onde  $(n_{i-1} + 1)$  é o número de neurônios da camada anterior, que define o número de entradas da camada atual, mais o viés do

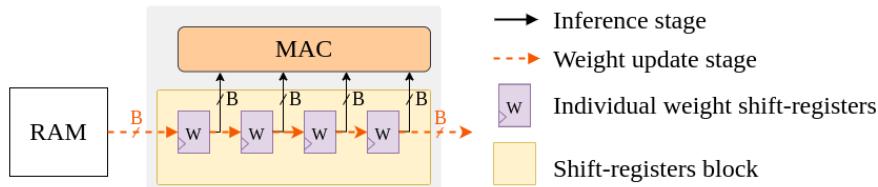


Figura 23 – Shift-register structure.

neurônio. Os  $N + 1$  representam um peso para cada uma das  $N$  entradas, mais o viés do neurônio. Os  $L + 1$  ciclos extras representam a camada de saída.

Uma vez que os pesos são carregados, o modo de inferência pode ser realizado. A quantidade de ciclos depende da implementação da multiplicação-adição acumulada (MAC, na sigla em inglês) utilizada (MAC de ciclo único ou pipeline). Para MACs de ciclo único, a etapa de inferência requer ( $L + 1$ ) ciclos.

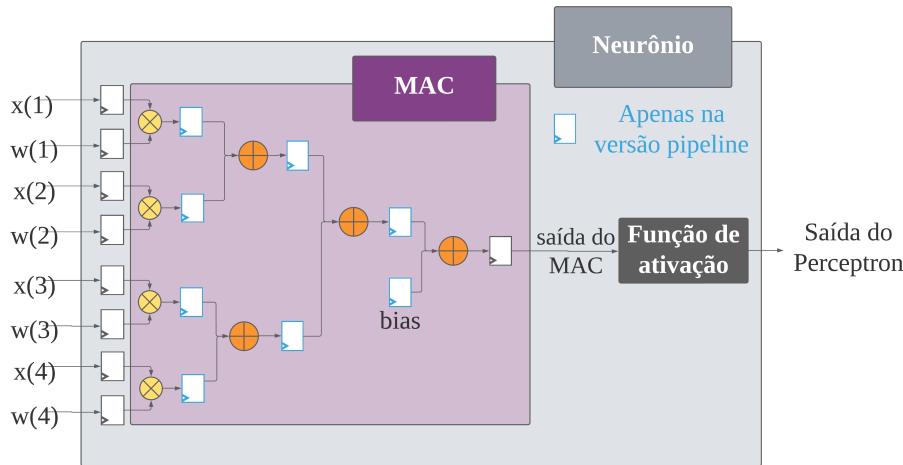
Para MACs de pipeline, o número de nós ( $n$ ) em cada camada também faz parte da equação, e o número de ciclos é obtido por  $(N + 1) + \sum_{i=2}^{L+1} (\log_2(n_{i-1}) + 1)$ .

### 5.1.5 Modularização das Arquiteturas

Uma das principais vantagens do framework é a modularização das arquiteturas geradas. Cada camada e unidade aritmética possui um nome de identificação, o que permite ao usuário alterar facilmente as arquiteturas conforme necessário. Essa modularidade simplifica a realização de testes rápidos com a criação de arquiteturas próprias, substituindo os módulos existentes, pois o usuário pode focar em uma determinada unidade ou camada, sem afetar o restante da rede neural. Essa flexibilidade facilita a exploração de diferentes configurações arquiteturais e promove um processo de projeto iterativo muito mais rápido e eficiente.

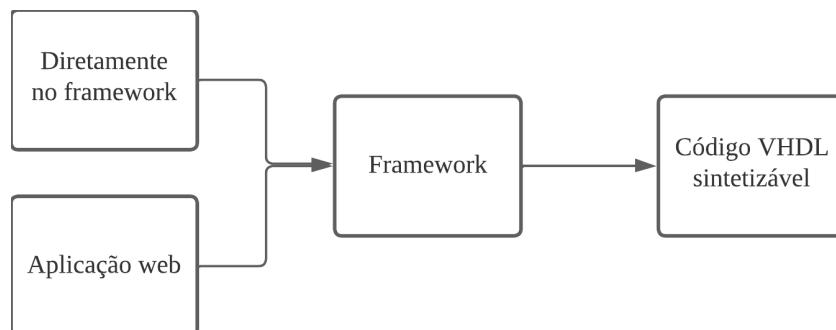
O fluxo de geração HDL pode ser visto na Figura 25. Diferentemente dos frameworks existentes abordados no Capítulo 4, que requerem a instalação de várias ferramentas e softwares licenciados, o framework desenvolvido neste trabalho oferece uma interface web intuitiva. Essa interface permite que o usuário ajuste facilmente as configurações desejadas para a geração do código VHDL, tornando todo o processo rápido e acessível. Com apenas alguns minutos de configuração, o usuário pode obter o código VHDL completo para a rede neural desejada.

Figura 24 – Arquitetura genérica de um Neurônio (exemplo para arquitetura com 4 entradas).



Fonte: autor.

Figura 25 – Fluxograma do framework



Fonte: autor.

### 5.1.6 Resultados e Limitações

Embora o framework tenha atingido seus objetivos principais e demonstrado várias capacidades e facilidades para o projeto de redes neurais artificiais em VHDL, é importante reconhecer suas limitações. Devido à restrição de tempo para a entrega deste trabalho, os testes para verificar a funcionalidade completa das arquiteturas geradas não puderam ser realizados. Portanto, é necessário realizar testes e validações adicionais para garantir o correto funcionamento das redes geradas pelo framework em diferentes cenários. Outro ponto é que o framework não possui um conversor direto dos modelos mais conhecidos como Tensorflow, Pytorch ou ONNX por exemplo, necessitando ao usuário atualmente, obter os pesos e viéses da arquitetura, para salvar em arquivo para

leitura e atualização dos valores na arquitetura em hardware gerada.

## 5.2 Resultados de Síntese

Esta seção apresenta dois conjuntos de experimentos: o primeiro revela a relação entre os parâmetros do framework e sua influência nos resultados da síntese de hardware. Ao variar esses parâmetros, pretendemos elucidar os efeitos consequentes em métricas críticas, como área, atraso e consumo de energia. A segunda análise avalia como esses parâmetros, combinados com o uso da aritmética de ponto fixo, afetam o desempenho do modelo.

### 5.2.1 Resultados de Síntese

Todas as arquiteturas foram sintetizadas considerando um fluxo de design ASIC para uma biblioteca de células padrão de 65nm da ST Micron, usando a ferramenta Cadence Genus. A frequência alvo foi definida como 200 MHz para todos os projetos. Os resultados de consumo de energia foram estimados com os valores padrão de atividade de comutação da ferramenta. Além disso, o ModelSim foi utilizado para realizar a verificação funcional dos circuitos.

Para mostrar um exemplo prático de nosso framework, várias arquiteturas de redes neurais (geradas automaticamente) são comparadas. As arquiteturas foram geradas com diferentes valores de precisão ( $B$  bits), paralelismo de entrada ( $N$  entradas e pesos), número de neurônios por camada ( $n$ ) e número de camadas ( $L$ ). Alguns projetos requeriam vários minutos para serem sintetizados devido ao tamanho do circuito, então mantivemos alguns parâmetros constantes, a saber, tamanho da camada de entrada/saída (16 e 10), implementação do MAC (ciclo único) e função de ativação (ReLU).

Tabela 1 – Design parameters tested (default values in bold).

Parameter	Values
MAC unit	Single-cycle
activation	ReLU
input layer size (N)	16
output layer size	10
input width (B)	4, <b>6</b> , 8
nb. of neurons per layer (n)	<b>4</b> , 8, 16
hidden layers (L)	<b>1</b> , 2, 4

As configurações testadas (um total de 9) são apresentadas na Tabela 1. Quando mais de um valor foi testado para um determinado parâmetro, os demais foram mantidos com os valores indicados em negrito.

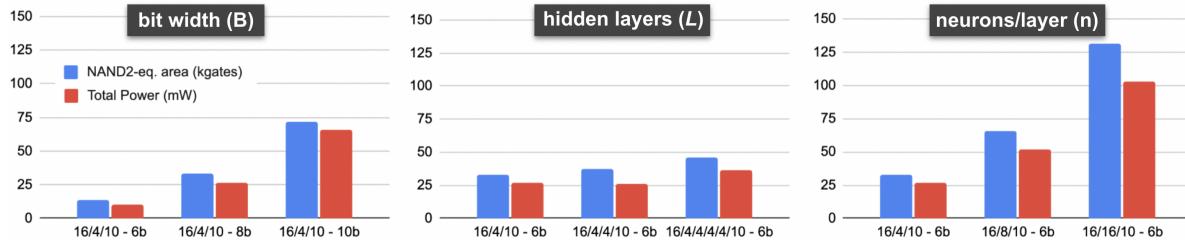


Figura 26 – Resultados de síntese para a frequência alvo de 200 MHz.

A Fig. 26 mostra os resultados de síntese para os parâmetros testados em termos de área (equivalente a NAND-2) e dissipação total de energia. Cada parâmetro é discutido nos parágrafos seguintes.

**Largura de bits:** Aumentar a largura de bits em um único incremento resulta em um aumento substancial na área de 14.525 kgates e um aumento significativo na potência de 13.975 mW. Ao dobrar a largura de bits, a área se multiplica por um impressionante fator de 5.368x, e a potência aumenta substancialmente em 6.534x. Isso enfatiza o trade-off entre precisão (largura de bits) e utilização de recursos de hardware, onde larguras de bits mais altas demandam consideravelmente mais área e potência.

**Camadas ocultas:** A adição de cada camada oculta contribui com 4.3 kgates para a área total e 5.05 mW para o consumo de potência. Ao dobrar o número de camadas ocultas, a área experimenta um aumento modesto de 1,23x no máximo, enquanto o consumo de potência aumenta em um máximo de 1,38x. Esses resultados sugerem que o impacto das camadas ocultas na área e na potência permanece relativamente limitado, mesmo com incrementos significativos de camadas, mas é importante observar que esses resultados são intrinsecamente influenciados pelo número de neurônios por camada que escolhemos manter.

**Neurônios por camada:** Cada incremento no número de neurônios por camada resulta em um acréscimo de 8.262,5 kgates na área e 6.337 mW na potência. Dobrar o número de neurônios por camada leva a um aumento máximo na área de 2,01x e um aumento na potência de 1,97x. Essas descobertas destacam que o número de neurônios por camada tem um crescimento quase linear com a área e a potência.

### 5.2.2 Efeitos da Aproximação em Redes Neurais de Classificação

Para evitar unidades aritméticas de ponto flutuante, os multiplicadores são implementados usando aritmética de ponto fixo. Como isso introduz um erro em nosso cálculo MAC, foram testados diferentes níveis de precisão.

Porém existe um preço a ser pago em termos de desempenho do modelo quando multiplicadores de ponto fixo são utilizados. No entanto, é difícil avaliar essa perda porque as bibliotecas de software para treinamento de redes neurais trabalham com precisão de ponto flutuante e não oferecem alternativas de ponto fixo.

Para avaliar o impacto dessas adaptações na acurácia dos modelos, desenvolveu-se um algoritmo em Python, liderado pelo aluno Lucas Soares, com o apoio do professor coorientador Mateus Grelert. Este algoritmo tem a capacidade de transformar um modelo previamente treinado, armazenado em formato '.h5', em um modelo adaptado. Ele realiza essa adaptação ajustando os pesos para a representação em ponto-fixo, com a possibilidade de definir a largura de bits desejada e, ao mesmo tempo, imita a arquitetura original, alterando a aritmética para simular o comportamento da arquitetura gerada em hardware.

Através deste processo, o algoritmo é capaz de realizar operações de inferência utilizando a aritmética adaptada e registra os resultados de saída do modelo adaptado. Essa abordagem possibilita o cálculo da acurácia do modelo adaptado, fornecendo revelações valiosas sobre como as adaptações impactam o desempenho do modelo, dependendo de cada configuração de arquitetura.

O estudo concentrou-se em um modelo de classificação que utiliza o conjunto de dados MNIST (DENG, 2012) com 10 classes. Os conjuntos de treinamento e teste contêm, respectivamente, 60.000 e 10.000 amostras. Devido às limitações de tamanho do netlist, tivemos que reduzir o tamanho da imagem de entrada de 28x28 para 4x4, o que prejudicou significativamente o desempenho. No entanto, isso não afeta o objetivo principal desta análise, que se concentra na comparação entre modelos de rede precisos e aproximados. Nossa configuração utilizou Python 3.9.5, com 30 épocas, uma taxa de aprendizado de 0,001, otimizador Adam e um tamanho de lote de 32. A comparação foi feita variando o número de camadas, mantendo o número de neurônios em cada camada ( $n$ ) em 4. Também é importante destacar que as redes neurais de ponto flutuante (referência) foram convertidas para ponto fixo após o treinamento, de modo que a aproximação está presente apenas na fase de inferência. Os resultados são apresentados na Figura 27.

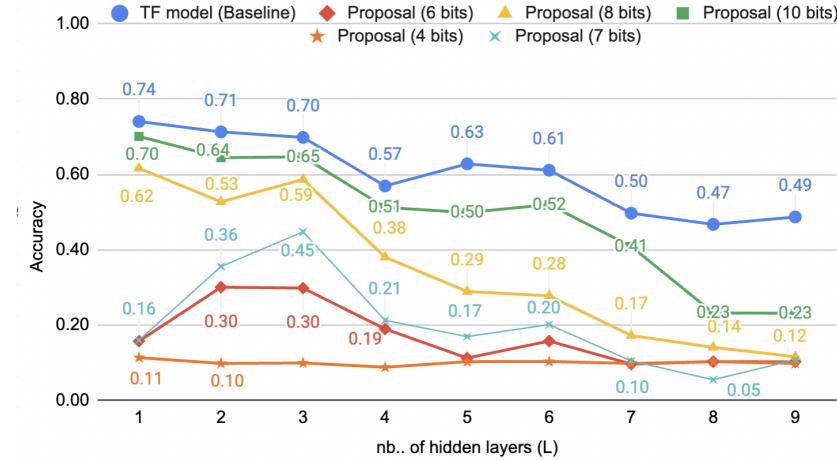


Figura 27 – Desempenho na inferência de classificação considerando a variação do número total de camadas (N) e bits de precisão (B) (imagem 4x4).

Os resultados da Figura 27 mostram que os modelos quantizados apresentam desempenho variado, especialmente quando menos bits são usados para representar os dados. Quando são utilizados 8 ou 10 bits, o desempenho do modelo aproximado está relativamente próximo da melhor versão precisa (que apresentou a melhor precisão de 74%), atingindo picos de desempenho de 70% e 62% para os modelos de 10 bits e 8 bits, com uma única camada oculta. No entanto, a diferença aumenta à medida que mais camadas ocultas são usadas. Isso pode parecer contra-intuitivo à primeira vista, mas pode ser resultado de overfitting ou porque mais épocas de treinamento são necessárias (já que mais camadas se traduzem em mais parâmetros à serem treinados).

Outro resultado interessante evidencia a necessidade de equilíbrio entre tamanho de arquitetura e desempenho do modelo. De forma à se economizar em área e consumo de potência, é natural a busca por modelos mais enxutos (imagens de treinamento menores, menos entradas, menos neurônios e menor representação de bits). Porém quando levado isso ao extremo, compromete a acurácia do modelo. Agora reduzindo o tamanho da imagem de entrada de 28x28 para 8x8 (ao invés de 4x4), e alterando o número de neurônios por camada oculta em 64 (ao invés de 4) o desempenho dos modelos foi significativamente melhor. Conforme a Figura 28, mesmo um modelo com representação de 6 bits apresenta um desempenho relativo satisfatório comparado ao modelo original (para um número de até 6 camadas ocultas), ou então um modelo de 8 bits para todos os números de camadas ocultas testados.

Em conclusão, nossas análises de regressão forneceram insights valiosos sobre a influência dos principais parâmetros de design de redes neurais nas métricas de síntese de

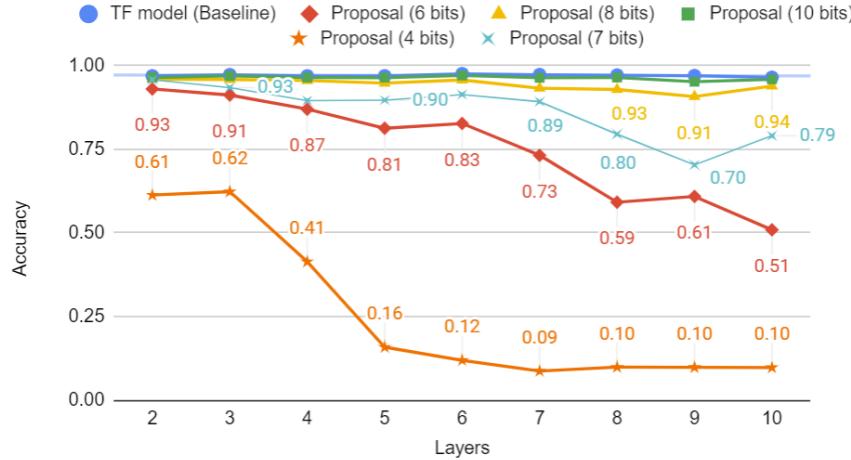


Figura 28 – Desempenho na inferência de classificação considerando a variação do número total de camadas ( $N$ ) e bits de precisão ( $B$ ) (imagem 8x8).

hardware e sua correlação com o desempenho da precisão do modelo.

O parâmetro de largura de bits emerge como o fator mais influente, com implicações substanciais em termos de área e consumo de energia. Maior precisão exige significativamente mais recursos, destacando o trade-off entre largura de bits e utilização de hardware.

Quanto à precisão do modelo, os modelos quantizados têm desempenho próximo à versão de maior precisão com 8 ou 10 bits. No entanto, a introdução de mais camadas ocultas amplia a lacuna de desempenho (resultado esperado devido ao erro propagado por mais camadas), sugerindo também desafios potenciais relacionados ao overfitting ou à duração do treinamento.

### 5.3 Melhorias para arquiteturas geradas pelo *Framewor*k

Os resultados do estudo descrito nas próximas 2 seções se concreta em um número de 4 a 20 vetores de entrada para ambas arquiteturas, pois o algoritmo Spiral com interface web possui a limitação de no máximo 20 constantes de multiplicação. Porém este número está longe de um número de vetores em uma aplicação real. Arquiteturas com aplicações reais costumam passar em muito deste número, como por exemplo a arquitetura MLP-Mixer, que possui blocos de camadas MLPs completamente conectadas 196 vetores de entrada, (TOLSTIKHIN et al., 2021).

Número de vetores	Área total	Área total	Variação
	arquitetura A	arquitetura B	
4	7800	8661	11.04%
8	13172	11802	-10.40%
12	23720	20754	-12.50%
16	33060	27153	-17.87%
20	40117	34865	-13.09%

Tabela 2 – Fonte: autor.

É importante salientar que devido aos vetores de entrada de cada neurônio, que representam suas entradas e pesos respectivos possuírem uma largura  $B$  de representação, os vetores resultantes da etapa de multiplicação (mostrados na seção 5.3.1) possuem uma largura  $2 * B$ , ou seja, para uma *MLP* com 8 bits de representação para suas entradas e saídas, possuirá valores destes vetores intermediários com 16 bits de largura.

### 5.3.1 Substituição dos Multiplicadores por Blocos de Somas e Deslocamentos

Na seção 4.2.1 deste estudo, foi detalhada a comparação entre duas arquiteturas diferentes, a arquitetura A, que usa multiplicadores padrão, e a arquitetura B, que substitui esses multiplicadores por blocos de somas e deslocamentos usando a ferramenta Spiral (Figura 17). Foram analisadas três métricas principais: área total, atraso crítico e potência total e avaliada a variação entre as duas arquiteturas.

Com relação à área total, observamos que a arquitetura B, com os blocos de somas e deslocamentos, teve um acréscimo de 11,04% em relação à arquitetura A para 4 vetores. No entanto, à medida que o número de vetores aumenta, a arquitetura B apresentou uma redução significativa na área necessária. Para 20 vetores, a arquitetura B apresentou uma redução de 13,09% em relação à arquitetura A. Esses resultados indicam que a substituição dos multiplicadores por somas e deslocamentos pode resultar em uma diminuição geral da área ocupada pela arquitetura, principalmente para maior quantidade de vetores, ou seja, para MLPs com maior número de neurônios por camada.

Em relação ao atraso crítico, olhamos para o tempo necessário para um sinal percorrer o caminho mais longo na arquitetura. Descobrimos que a Arquitetura B experimentou aumentos significativos no atraso crítico em comparação com a Arquitetura A. Para 4 vetores, observamos um aumento de 42,40% em um atraso crítico na arquitetura B.

Número de vetores	Atraso crítico arquitetura A	Atraso crítico arquitetura B	Variação
4	3064	4363	42.40%
8	3064	4232	38.12%
12	3528	4814	36.45%
16	3756	5246	39.67%
20	3714	5220	40.55%

Tabela 3 – Fonte: autor.

Número de vetores	Potência total arquitetura A	Potência total arquitetura B	Variação
4	411663.426	596617.265	44.93%
8	708515.042	731622.504	3.26%
12	1321226.936	1537422.441	16.36%
16	1883624.351	2105948.442	11.80%
20	2281310.749	2622236.593	14.94%

Tabela 4 – Fonte: autor.

O ganho proporcional se manteve próximo deste valor mesmo quando o número de vetores aumentou, ou seja, a Arquitetura B mostrou atrasos maiores em comparação com a Arquitetura A em todos os casos. Esses resultados sugerem que a substituição de multiplicadores por somas impactará o desempenho da arquitetura em termos de atraso crítico, o que sugere uma troca entre atraso crítico e área total.

Os resultados gerados demonstram que a Arquitetura B, com os blocos de adição e deslocamento, apresentou aumentos na potência total em comparação com a Arquitetura A. Para 4 vetores, a arquitetura B apresentou um aumento de 44,93% na potência total. No entanto, à medida que o número de vetores aumentou, a diferença na potência total entre as duas arquiteturas diminui, reforçando o maior benefício do uso desta técnica para arquiteturas MLPs de maior número de neurônios. Em todos os casos, a arquitetura B apresentou maior potência total comparada à arquitetura A. Esses resultados indicam que a substituição dos multiplicadores por somas e deslocamentos resulta em um aumento geral no consumo de energia da arquitetura.

Em resumo, os resultados constatam que a substituição de multiplicadores por blocos de somas e deslocamentos, utilizando a ferramenta Spiral, traz impactos positivos na ocupação de área total porém ao custo de impactos negativos no atraso crítico e consumo de potência. No geral, utilizar técnicas de somas e deslocamentos se mostra válida para

arquiteturas em que a área ocupada seja um fator crítico. Esses resultados destacam a importância de considerar cuidadosamente as compensações entre essas análises ao escolher substituir multiplicadores por somas e mudanças em uma arquitetura de rede neural, dependendo do objetivo do projeto.

### 5.3.2 Troca dos Somadores por um Bloco de Compressão de Vetores

A pesquisa teve como objetivo comparar dois tipos de árvore de soma em hardware considerando diferentes valores de largura de bits  $B$  e número de vetores  $N$ . Os dois tipos de árvore de soma foram denominados **tipo 1** e **tipo 2**. O **tipo 1** (exemplificado na Figura 16b), realizou a soma direta dos vetores utilizando os operadores de soma '+' no código de descrição de hardware (*HDL*), enquanto o **tipo 2** (exemplificado na Figura 18) utilizou a técnica de compressão dos vetores através de *Carry-Save Adders* (*CSAs*), conforme seção 4.2.2.

N	12 bits		16 bits		20 bits	
	Tipo 1	Tipo 2	Tipo 1	Tipo 2	Tipo 1	Tipo 2
4	4375	4343	5853	5816	7333	7296
8	10101	10052	13501	13451	16900	16850
12	15829	15581	21148	20900	26467	26219
16	21588	21536	28826	28774	36065	36013
20	27321	27333	36480	36371	45638	45410

Tabela 5 – Comparação de Área total entre arquitetura padrão (tipo 1) e compressores (tipo 2) - Fonte: autor.

Para diferentes valores de  $B$  (6, 8 e 10), conforme descrito na seção 5.3 trazem resultados intermediários com o dobro deste número de bits (12, 16 e 20).

Em comparação com o **tipo 1**, o uso da arquitetura do **tipo 2** apresentou uma pequena redução na área total de até 1,59%, 1,19% e 0,95% para para  $B = 6, 8$  e 10 bits. Isso indica que a técnica de compressão dos vetores utilizando *Carry-Save Adders* (*CSAs*) traz uma economia muito sutil na área total do circuito, principalmente para arquiteturas com menor número de bits, o que não justifica o uso da mesma.

A arquitetura do tipo 2 apresentou aumento de até 27,26% no atraso crítico para  $B = 6, N = 16$  e até 20,19% para  $B = 10, N = 16$ , o que sugere que o aumento da largura de representação  $B$  torna a utilização da arquitetura do tipo 2 mais atrativa, pois a diferença tende a diminuir a medida que  $B$  aumenta. O aumento no número de

N	12 bits		16 bits		20 bits	
	Tipo 1	Tipo 2	Tipo 1	Tipo 2	Tipo 1	Tipo 2
4	3446	3748	4434	4393	5421	5381
8	3967	4542	4955	5529	5942	6517
12	4437	4610	5425	5598	6412	6586
16	4637	5249	5625	6237	6640	7225
20	5096	5261	6084	6248	7071	7269

Tabela 6 – Comparaçāo de Atraso māximo entre arquitetura padrāo (tipo 1) e compressores (tipo 2) - Fonte: autor.

N	12 bits		16 bits		20 bits	
	Tipo 1	Tipo 2	Tipo 1	Tipo 2	Tipo 1	Tipo 2
4	401702.457	401728.741	553990.823	545848.085	661804.691	687322.391
8	1297960.433	1397620.704	1793935.518	1934067.116	2243942.746	2361437.367
12	2425587.731	1943495.959	3389768.361	2688477.451	4283689.599	3283983.452
16	3860425.249	4041805.554	5317028.93	5515710.797	6704144.651	6959152.4
20	5172204.588	5621586.712	7012814.39	7458084.027	8742118.715	9448382.084

Tabela 7 – Comparaçāo de Potênciā total entre arquitetura padrāo (tipo 1) e compressores (tipo 2) - Fonte: autor.

vetores de entrada  $N$  parece trazer resultados nāo conclusivos, uma vez que os mesmos variam bastante, possuindo o tipo 2 valores de atraso crítico menores apenas para  $N = 4$ .

Os resultados de potênciā total mostraram os melhores resultados para o tipo 2 com o nāmero de vetores  $N = 12$ , com valores 30,44% menores do que a arquitetura do tipo 1. Porém estes resultados promissores nāo se repetem para outros nāmeros de vetores de entrada, o que traz uma indicação de que existem nāmeros de vetores ótimos que tornam a arquitetura tipo 2 mais atrativa. Os dois melhores nāmeros de  $N$  vetores se revelam sendo para  $N = 4$  e  $N = 12$ , dentro do alcance estudado neste trabalho.

A pesquisa comparativa entre os dois tipos de árvore de soma em hardware, **tipo 1** e **tipo 2**, revelou que a utilização de Carry-Save Adders para compressão dos vetores nāo apresenta vantagens significativas em termos de área total do circuito. Independentemente da largura de bits utilizada, o **tipo 2** resultou em uma área total sutilmente menor em comparação com o **tipo 1**, ao passo de resultados consideravelmente piores no atraso crítico e consumo de potênciā. Em teoria compressores de soma deveriam ser muito mais eficientes em termos de consumo de área, porém as otimizações de síntese da ferramenta utilizada demonstra já fazer uma série de otimizações, nāo compensando o uso de compressores de soma ao invés dos operadores '+'.

## 6 Conclusão

No âmbito deste Trabalho de Conclusão de Curso (TCC) em Engenharia Eletrônica, desenvolveu-se um framework em Python voltado para a geração automatizada de código VHDL sintetizável destinado a aceleradores de Perceptrons de Múltiplas Camadas, do inglês "Multi-Layer Perceptrons"(MLPs). O trabalho abrangeu três principais áreas de pesquisa que contribuíram significativamente para o avanço na otimização dessas arquiteturas de hardware.

No que diz respeito à criação do framework, observou-se que a largura de bits dos dados tem um impacto substancial na área e no consumo de energia. Aumentar a largura de bits resulta em um aumento considerável tanto na área quanto na potência, enfatizando um claro trade-off entre precisão (largura de bits) e a utilização de recursos de hardware. A adição de camadas ocultas teve um impacto relativamente limitado em termos de área e potência, mesmo com incrementos significativos no número de camadas. No entanto, é importante considerar que esses resultados são influenciados pelo número de neurônios por camada escolhido, sendo o correto equacionamento deste impacto, o número total de multiplicadores e somadores na arquitetura.

No estudo sobre os efeitos da aproximação em redes neurais de classificação, ficou evidente mais uma vez que a largura de bits é o fator mais influente, com implicações substanciais em termos de área e consumo de energia. Modelos quantizados apresentaram desempenho próximo ao da versão de maior precisão com 8 ou 10 bits. No entanto, a introdução de mais camadas ocultas ampliou a lacuna de desempenho, sugerindo desafios relacionados ao erro propagado, overfitting e/ou à duração do treinamento.

O último estudo focou-se em possíveis otimizações na arquitetura do acelerador de MLPs. Duas abordagens foram comparadas: a substituição de multiplicadores por blocos de somas e deslocamentos, bem como a troca de somadores por um bloco de compressão de vetores. Constatou-se que a primeira abordagem, embora otimize a área ocupada, implica em impactos negativos no atraso crítico e no consumo de potência. Portanto, essa técnica se mostra adequada para situações em que a área é o fator crítico. Quanto à comparação entre os dois tipos de árvore de soma em hardware, a utilização de Carry-Save Adders para compressão dos vetores não apresentou vantagens significativas em termos de área total do circuito, resultando em resultados consideravelmente piores em atraso crítico e consumo de potência.

Em resumo, este trabalho proporcionou um estudo significativo na otimização de

aceleradores de MLPs em hardware por meio da criação de um framework de geração de código VHDL, do estudo dos efeitos da aproximação na acurácia do modelo e da análise de possíveis otimizações na arquitetura do acelerador. Essas descobertas contribuem para a compreensão mais profunda dos trade-offs envolvidos na adaptação de modelos de redes neurais para implementações em hardware e fornecem intuições valiosas para futuros desenvolvimentos na área.

Para pesquisas futuras, sugerem-se as seguintes direções:

Desenvolvimento de uma ferramenta de conversão automática capaz de transformar modelos em formatos comuns, como ONNX, PyTorch e TensorFlow, em descrições em VHDL. Isso ampliaria significativamente a utilidade do framework desenvolvido, facilitando a adaptação de uma variedade de modelos de redes neurais para implementações em hardware.

Exploração da substituição dos MACs (Multiply-Accumulate Units) combinacionais por blocos sequenciais. Essa mudança poderia impactar o desempenho, consumo de área, potência e atraso das arquiteturas de hardware. Investigar os trade-offs envolvidos nessa transição seria um campo promissor de pesquisa.

Considerando o amplo uso das camadas convolucionais em Redes Neurais Convolucionais (CNNs) para aplicações de visão e áudio, uma área de pesquisa interessante seria a geração automatizada de camadas convolucionais. Isso contribuiria para a expansão do framework para suportar uma variedade mais ampla de arquiteturas de redes neurais, abrangendo domínios além de MLPs.

# Referências Bibliográficas

ACADEMY, D. S. *Capítulo 58 - Introdução aos Autoencoders - Deep Learning Book*. 2022. <<https://www.deeplearningbook.com.br/introducao-aos-autoencoders/>>. (Accessed on 12/09/2022). 24

AMORIM, P. W. F. *Autoencoder*. 2022. Disponível em: <<https://lamfo-unb.github.io/2020/11/21/Autoencoder/>>. 24

AWATI, R. *What is an intellectual property core (IP core)? – techtarget definition*. TechTarget, 2022. Disponível em: <<https://www.techtarget.com/whatis/definition/IP-core-intellectual-property-core>>. 18

BRE, F.; GIMENEZ, J. M.; FACHINOTTI, V. D. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, v. 158, p. 1429–1441, 2018. ISSN 0378-7788. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0378778817325501>>. 22

COLUCCI, A. et al. A fast design space exploration framework for the deep learning accelerators: Work-in-progress. In: *2020 International Conference on Hardware/Software Codesign and System Synthesis (CODESISSS)*. IEEE, 2020. Disponível em: <<https://doi.org/10.1109/codesisss51650.2020.9244038>>. 18

DENG, B. L. et al. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, Institute of Electrical and Electronics Engineers (IEEE), v. 108, n. 4, p. 485–532, abr. 2020. Disponível em: <<https://doi.org/10.1109/jproc.2020.2976475>>. 16

DENG, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, IEEE, v. 29, n. 6, p. 141–142, 2012. 41

DESROUSSEAUX, R.; BERNARD, G.; MARIAGE, J.-J. Profiling money laundering with neural networks: a case study on environmental crime detection. In: *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.: s.n.], 2021. p. 364–369. 11

DOBILAS, S. *Autoencoders (AE) — A Smart Way to Process Your Data Using Unsupervised Neural Networks*. 2022. Disponível em: <<https://towardsdatascience.com/autoencoders-ae-a-smart-way-to-process-your-data-using-unsupervised-neural-networks-9661f93a8509#5a19-bd0fb45c9472>>. 21

DSA, E. *Capítulo 58 - Introdução AOS autoencoders*. 2019. Disponível em: <<https://www.deeplearningbook.com.br/introducao-aos-autoencoders/>>. 24

- FAHIM, F. et al. *hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices*. arXiv, 2021. Disponível em: <<https://arxiv.org/abs/2103.05579>>. 27
- GUO, K. et al. Software-hardware codesign for efficient neural network acceleration. *IEEE Micro*, Institute of Electrical and Electronics Engineers (IEEE), v. 37, n. 2, p. 18–25, mar. 2017. Disponível em: <<https://doi.org/10.1109/mm.2017.39>>. 14, 15
- GUO, K. et al. [DL] a survey of FPGA-based neural network inference accelerators. *ACM Transactions on Reconfigurable Technology and Systems*, Association for Computing Machinery (ACM), v. 12, n. 1, p. 1–26, mar. 2019. Disponível em: <<https://doi.org/10.1145/3289185>>. 11
- GYAN, V. *Carry save adder verilog code: Verilog implementation of carry save adder*. Admin, 2022. Disponível em: <<http://vlsigyan.com/carry-save-adder-verilog-code/>>. 33
- HABIBIAN, A. et al. Video compression with rate-distortion autoencoders. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2019. p. 7032–7041. 24
- HAN, S. et al. EIE: Efficient inference engine on compressed deep neural network. In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016. Disponível em: <<https://doi.org/10.1109/isca.2016.30>>. 7, 12, 14
- HAO, C.; CHEN, D. Deep neural network model and FPGA accelerator co-design: Opportunities and challenges. In: *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE, 2018. Disponível em: <<https://doi.org/10.1109/icsict.2018.8564956>>. 12
- HENNESSY, J. L.; PATTERSON, D. A.; ASANOVIC, K. *Computer Architecture: A quantitative approach*. [S.l.]: Morgan Kaufmann, 2019. 16
- HINKELMANN, P. D. K. *Wayback Machine*. 2022. <[https://web.archive.org/web/20181006235506/http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay\\_1718:ke-11\\_neural\\_networks.pdf](https://web.archive.org/web/20181006235506/http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf)>. (Accessed on 12/09/2022). 23
- IBERDROLA.COM. *Conheça os principais benefícios do 'Machine Learning'*. 2022. Disponível em: <<https://www.iberdrola.com/inovacao/o-que-e-machine-learning>>. 20
- ISELE, D. et al. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018. Disponível em: <<https://doi.org/10.1109/icra.2018.8461233>>. 11
- JADON, S. *Introduction to Different Activation Functions for Deep Learning / by Shruti Jadon / Medium*. 2018. <<https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092>>. (Accessed on 12/09/2022). 23

- JORDAN, J. *Introduction to autoencoders*. Jeremy Jordan, 2018. Disponível em: <<https://www.jeremyjordan.me/autoencoders/>>. 24
- JOUPPI, N. *Quantifying the performance of the TPU, our first machine learning chip / Google Cloud Blog*. 2017. <<https://cloud.google.com/blog/products/gcp/quantifying-the-performance-of-the-tpu-our-first-machine-learning-chip>>. (Accessed on 12/09/2022). 12, 13
- K, M. et al. 1d convolution approach to human activity recognition using sensor data and comparison with machine learning algorithms. *International Journal of Cognitive Computing in Engineering*, Elsevier BV, v. 2, p. 130–143, jun. 2021. Disponível em: <<https://doi.org/10.1016/j.ijcce.2021.09.001>>. 11
- LAI SUBUTAI AHMAD, D. D. C.; MAVER, C. *AI is harming our planet: addressing AI's staggering energy cost*. 2022. <<https://www.numenta.com/blog/2022/05/24/ai-is-harming-our-planet/>>. (Accessed on 12/09/2022). 13
- LEBEDEV, M.; BELECKY, P. A survey of open-source tools for FPGA-based inference of artificial neural networks. In: *2021 Ivannikov Memorial Workshop (IVMEM)*. IEEE, 2021. Disponível em: <<https://doi.org/10.1109/ivmem53963.2021.00015>>. 25
- LIU, X. et al. Compressed ultrahigh-speed photography enabled by a snapshot-to-video autoencoder. In: *2022 Photonics North (PN)*. IEEE, 2022. Disponível em: <<https://doi.org/10.1109/pn56061.2022.9908322>>. 16
- LU, W. et al. Secure robust jpeg steganography based on autoencoder with adaptive bch encoding. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 31, n. 7, p. 2909–2922, 2021. 11
- MISRA, J.; SAHA, I. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, Elsevier BV, v. 74, n. 1-3, p. 239–255, dez. 2010. Disponível em: <<https://doi.org/10.1016/j.neucom.2010.03.021>>. 14
- MORGAN, L. *AI carbon footprint: Helping and hurting the environment / TechTarget*. 2021. <<https://www.techtarget.com/searchenterpriseai/feature/AI-carbon-footprint-Helping-and-hurting-the-environment>>. (Accessed on 12/09/2022). 12
- Multilayer perceptron. *Multilayer perceptron — Wikipedia, The Free Encyclopedia*. 2022. [Online; accessed 28-December-2022]. Disponível em: <[https://en.wikipedia.org/wiki/Multilayer\\_perceptron](https://en.wikipedia.org/wiki/Multilayer_perceptron)>. 29
- NASCIMENTO, R. S. F. do et al. Detecção de anomalias em poços de petróleo surgentes com stacked autoencoders. In: *Proceedings do XV Simpósio Brasileiro de Automação Inteligente*. SBA Sociedade Brasileira de Automática, 2021. Disponível em: <<https://doi.org/10.20906/sbai.v1i1.2856>>. 24
- NURVITADHI, E. et al. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In: *2016 26th International Conference*

- on Field Programmable Logic and Applications (FPL). IEEE, 2016. Disponível em: <<https://doi.org/10.1109/fpl.2016.7577314>>. 12, 15
- OUYANG, Z. et al. Deep CNN-based real-time traffic light detector for self-driving vehicles. *IEEE Transactions on Mobile Computing*, Institute of Electrical and Electronics Engineers (IEEE), v. 19, n. 2, p. 300–313, fev. 2020. Disponível em: <<https://doi.org/10.1109/tmc.2019.2892451>>. 11
- PAN, Y. Heading toward artificial intelligence 2.0. *Engineering*, v. 2, n. 4, p. 409–413, 2016. ISSN 2095-8099. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2095809917300772>>. 11
- PESSOA, J. et al. End-to-end learning of video compression using spatio-temporal autoencoders. In: *2020 IEEE Workshop on Signal Processing Systems (SiPS)*. [S.l.: s.n.], 2020. p. 1–6. 11
- PRECEDENCERESearch. *Artificial Intelligence Market*. 2022. Disponível em: <<https://www.precedenceresearch.com/artificial-intelligence-market>>. 11
- SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, v. 61, p. 85–117, 2015. ISSN 0893-6080. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0893608014002135>>. 11
- SEMIENGINEERING.COM. *RTL (Register Transfer Level)*. semiengineering.com, 2021. Disponível em: <[https://semiengineering.com/knowledge\\_centers/eda-design/definitions/register-transfer-level/](https://semiengineering.com/knowledge_centers/eda-design/definitions/register-transfer-level/)>. 25
- STRUBELL, E.; GANESH, A.; MCCALLUM, A. *Energy and Policy Considerations for Deep Learning in NLP*. arXiv, 2019. Disponível em: <<https://arxiv.org/abs/1906.02243>>. 12
- SZE, V. et al. *Efficient Processing of Deep Neural Networks: A Tutorial and Survey*. arXiv, 2017. Disponível em: <<https://arxiv.org/abs/1703.09039>>. 27
- TECNOBLOG, L. K. *O que É dma? [Acesso Direto à memória]*. tecnoblog.net, 2022. Disponível em: <<https://tecnoblog.net/responde/o-que-e-dma-acesso-direto-a-memoria/>>. 28
- THOMPSON, N.; SPANUTH, S. The decline of computers as a general purpose technology: Why deep learning and the end of moore's law are fragmenting computing. *SSRN Electronic Journal*, Elsevier BV, 2018. Disponível em: <<https://doi.org/10.2139/ssrn.3287769>>. 12, 13, 16
- TOLSTIKHIN, I. O. et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, v. 34, p. 24261–24272, 2021. 43
- WANG, J. et al. Neural RRT: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, Institute of Electrical and

Electronics Engineers (IEEE), v. 17, n. 4, p. 1748–1758, out. 2020. Disponível em: <<https://doi.org/10.1109/tase.2020.2976560>>. 11

WIKIPEDIA. *High-level synthesis - Wikipedia*. 2022. <[https://en.wikipedia.org/wiki/High-level\\_synthesis](https://en.wikipedia.org/wiki/High-level_synthesis)>. (Accessed on 12/09/2022). 27

WIKIPEDIA. *History of artificial intelligence*. 2022. Disponível em: <[https://en.wikipedia.org/wiki/History\\_of\\_artificial\\_intelligence#cite\\_note-178](https://en.wikipedia.org/wiki/History_of_artificial_intelligence#cite_note-178)>. 11

XILINX, I. *Vitis AI*. AMD - Advanced Micro Devices, Inc, 2022. (Accessed on 12/25/2022). Disponível em: <<https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html>>. 25

XILINX, I. *Xilinx Vitis AI documentation portal*. AMD - Advanced Micro Devices, AMD, 2022. Disponível em: <<https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Zynq-UltraScale-MPSOC-DPUCZDX8G>>. 26

YAMAZAKI. *NNgen/nngen: NNgen: A fully-customizable hardware synthesis compiler for deep neural network*. Shinya Takamaeda-Yamazaki and Contributors, 2022. Disponível em: <<https://github.com/NNgen/nngen>>. 28

YURTSEVER, E. et al. A survey of autonomous driving: icommon practices and emerging technologies/i. *IEEE Access*, Institute of Electrical and Electronics Engineers (IEEE), v. 8, p. 58443–58469, 2020. Disponível em: <<https://doi.org/10.1109/access.2020.2983149>>. 12

ZHANG, G. et al. Cnn-based sample adaptive offset optimization in hevc for streaming video. In: *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. [S.l.: s.n.], 2019. p. 263–266. 11

ZHOU, J.; LV, T.; YI, X. End-to-end distributed video coding. In: *2022 Data Compression Conference (DCC)*. [S.l.: s.n.], 2022. p. 496–496. 11

# Apêndices

# APÊNDICE A – Tabela comparativa de Frameworks

Tabela 8 – Tabela Comparativa de frameworks

Nome	Ano Public.	Ano Atuali.	Selecionado?	Comprime o modelo?	Treina modelo?	Ajuste fino automático dos parâmetros?	Otimização para Zynq-7000?	Configurações de arquitetura personalizáveis?	Faz verificação?	Sintetiza em ASIC?	Faz validação da sintese?	Estima potência?	Estima área?	Estima latência?	Código aberto?
bis4ml	2021	2021	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LeFlow	2018	2019	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
ONNC	2019	2020	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
NNgen	-	2022	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓
DnnWeaver	2016	2019	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓
Panda-hannin	2021	2022	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓
FP-DNN	2017	-	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
NVDLA	2018	2018	✗	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✓
yolowell	2019	2020	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Vitis AI (Xilinx)	2020	2022	✓	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗
OpenVINO (Intel Arria FPGA)	2022	2022	✗	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗