UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUIS EDUARDO PEREIRA MENDES

Aceleradores para Redes Neurais Vision Transformer (ViT) em Sistemas RISC-V avaliados em dispositivos de borda

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Antônio Carlos S. Beck

Porto Alegre
2026

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profª. Marcia Cristina Bernardes Barbosa
Vice-Reitor: Prof. Pedro de Almeida Costa
Pró-Reitora de Graduação: Profª. Nádya Pesce da Silveira
Diretor do Instituto de Informática: Prof. Luciano Paschoal Gaspary
Coordenador do Curso de Engenharia de Computação: Prof. Paulo Francisco Butzen
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro

“O universo não foi feito à medida do ser humano,
mas nós podemos tentar entendê-lo.“
— Carl Sagan

AGRADECIMENTOS

À minha mãe, meu pai, meus avós, minha irmã e toda minha família, que me apoiaram e confiaram em mim sempre. Aos amigos com quem eu sempre pude contar e às pessoas que estavam comigo durante esse período, àqueles que me confortaram e tornaram os momentos mais alegres. A todos os colegas de laboratório que sempre tornaram o ambiente de pesquisa divertido e descontraído. Aos professores e orientadores que não mediram esforços para auxiliar sempre, tanto no ensino e pesquisa como também na vida.

AGRADECIMENTOS INSTITUCIONAIS

Este trabalho foi parcialmente financiado pelo projeto AgroEdge suportado pelo Center for Embedded Devices and Research in Digital Agriculture (CEDRA) do SENAI-RS, com recursos financeiros oriundos do PPI IoT/Manufatura 4.0 / PPI HardwareBR do MCTI, através do Termo de Cooperação n.º 056/2023, firmado com a EMBRAPII, pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, pelo CNPq, pela FAPERGS Techfuturo - 23/2551-0002199-4, pelo Office of Naval Research Global (ONR) (N629092212014), e pelo PADIS através da HT Micron Semicondutores.

RESUMO

O desenvolvimento de aplicações de processamento de imagens para plataformas de computação de borda tem se mostrado uma abordagem valiosa nas mais diversas áreas do conhecimento. Nesse contexto, as Redes Neurais Convolucionais (CNNs) têm sido amplamente adotadas, mas estudos recentes demonstram a superioridade dos Vision Transformers (ViTs) em grandes volumes de dados, embora com um custo computacional mais elevado. Para otimizar a execução de modelos como os ViTs, surgiram coprocessadores de código aberto para a arquitetura RISC-V. Entre eles, destaca-se o Gemmini, parte do ecossistema Chipyard da Universidade de Berkeley. Embora eficiente para CNNs, a arquitetura do Gemmini não é otimizada para o fluxo de dados complexo dos ViTs, resultando em desempenho subótimo. Além disso, para acomodar modelos de grande escala em hardware com restrições de memória, a quantização é essencial para reduzir a utilização de memória do modelo e acelerar o processamento de matrizes, embora continue sendo um desafio para os Vision Transformers (ViTs) devido à distribuição não gaussiana de suas ativações. Para lidar com essas restrições, este trabalho apresenta a primeira implementação de ViTs no acelerador Gemmini. Oferecemos uma definição de modelo especializada, juntamente com uma estrutura altamente configurável e adaptada ao hardware para treinamento e quantização, otimizada especificamente para permitir a inferência eficiente de ViTs em sistemas baseados em Gemmini.

Palavras-chave: Vision transformers. ViT. arquitetura sistólica. ASIC. quantização. transferência de conhecimento. escalonamento arquitetural. dispositivos de borda.

Accelerators for Vision Transformer (ViT) Neural Networks on RISC-V Systems evaluated on Edge Devices

ABSTRACT

The development of image processing applications for edge computing platforms has proven to be a valuable approach in a wide range of fields. In this context, Convolutional Neural Networks (CNNs) have been widely adopted, but recent studies demonstrate the superiority of Vision Transformers (ViTs) when dealing with large volumes of data, at a higher computational cost. To optimize the execution of models such as ViTs, open-source coprocessors for the RISC-V architecture have emerged. Among them, Gemmini, which is part of the University of California, Berkeley's Chipyard ecosystem, stands out. Although efficient for CNNs, Gemmini's architecture is not optimized for the complex data flow of ViTs, resulting in suboptimal performance. Furthermore, to run large-scale models on hardware with memory constraints, quantization is essential for reducing the model's memory usage and accelerating matrix processing, although it remains a challenge for Vision Transformers (ViTs) due to the non-Gaussian distribution of their activations. To address these limitations, this paper presents the first implementation of ViTs on the Gemmini accelerator. We offer a specialized model definition, along with a highly configurable, hardware-tailored framework for training and quantization, optimized specifically to enable efficient ViT inference on Gemmini-based systems.

Keywords: vision transformers, ViT, systolic architecture, ASIC, quantization, transfer learning, architecture scaling, edge devices.

LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence
ASIC	Application Specific Integrated Circuit
CISC	Complex Instruction Set Computer
CNN	Convolutional Neural Network
DeiT	Data Efficient Image Transformer
DMA	Direct Memory Access
DNN	Deep Neural Networks
DRAM	Dynamic Random Access Memory
DSA	Domain Specific Architecture
DSE	Design Space Exploration
DSP	Digital Signal Processors
FCN	Fully-Connected Network
FFN	Feed-Forward Network
FP32	Floating Point 32-Bits
FPGA	Field Programmable Gate Array
GELU	Gaussian Error Linear Unit
GEMM	General Matrix Multiply
GEMV	General Matrix Vector Product
GPU	Graphic Processing Unit
INT8	Integer 8-Bits
IoT	Internet-of-Things
ISA	Instruction Set Architecture
MAC	Multiply-Accumulate
MHA	Multi Head Attention

MHSA	Multi Head Self-Attention
MLP	Multi Layer Perceptron
NLP	Natural Language Processing
NPU	Neural Processing Unit
OS	Output-Stationary
PE	Processing Elements
PTQ	Post-Training Quantization
QAT	Quantization Aware Training
ReLU	Rectified Linear Unit
RISC	Reduced Instruction Set Computer
RNN	Recurrent Neural Network
RoCC	Rocket Chip Coprocessor
SoC	System-on-Chip
Swin	Shifted Window Transformer
TPU	Tensor Processing Unit
ViT	Vision Transformer
WS	Weight-Stationary

LISTA DE FIGURAS

Figura 2.1 Tarefas de Computação Visual 19
Figura 2.2 Descrição por camadas da rede convolucional *alexnet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) 20
Figura 2.3 Topologia Vision Transformer (DOSOVITSKIY et al., 2021) 21
Figura 2.4 Processo de separação em *patches* e incorporação de posição (*Position Embedding*). (DOSOVITSKIY et al., 2021) 22
Figura 2.5 Topologia do mecanismo de Multi Head Self-Attention, (VASWANI et al., 2023) 23
Figura 2.6 Representação de uma FFN, ou rede Fully-Connected como uma operação de GEMV (JORDAN, 2017) 23
Figura 2.7 Destilação de Conhecimento 27
Figura 2.8 Distribuição da normalização Softmax e da função de ativação GELU (YUAN et al., 2021) 32
Figura 2.9 Microarquitetura do Arranjo Sistólico Gemmini (GENC et al., 2021) 33
Figura 2.10 System-on-Chip Gemmini (GENC et al., 2021) 34
Figura 2.11 Microarquitetura dos elementos de processamento (PE) dentro da hierarquia de 2 níveis, composta por *Tiles* de *PEs* 35

Figura 3.1 Comparação entre as diferentes implementações de Arranjo Sistólico (HUANG et al., 2023) 41
Figura 3.2 Arquitetura proposta pelo acelerador ViTA (NAG et al., 2023) 42

Figura 4.1 Grafo de fluxo de execução do framework desenvolvido 45
Figura 4.2 Aproximação de Schraudolph em comparação com o cálculo exato da exponencial 47
Figura 4.3 Pipeline de execução do Framework, partindo de um modelo estudante e um modelo professor pré-treinado, passando por um processo de *fine-tuning*, chegando até o modelo quantizado e funcional para o simulador Gemmini. 50
Figura 4.4 Esquema de quantização de precisão mista (W8A32) adotada 51

Figura 5.1 Imagem ilustrativa da placa Xilinx Nexys Video 54

Figura 6.1 Comparação entre os modelos MiniDeiT e MiniViT 60

LISTA DE TABELAS

Tabela 3.1 Comparação de características de Aceleradores para ViT recentes 43

Tabela 4.1 Dimensões estruturais dos modelos propostos com base na arquitetura MiniViT.....46
Tabela 4.2 Dimensões estruturais dos modelos propostos com base na arquitetura MiniDeiT.....46

Tabela 5.1 Características do Co-processador gemmini sintetizadas.....55

Tabela 6.1 Comparação entre a acurácia da execução dos modelos quantizados em INT8 (usando PTQ) contra a execução em FP32 na GPU.....58
Tabela 6.2 Comparação entre a dimensão dos modelos quantizados em INT8 comparado aos modelos FP32 na GPU.....59
Tabela 6.3 Comparação de Performance: Pesos Estacionários vs. CPU (MiniViT com CIFAR-10) 60
Tabela 6.4 Hardware Resource and Efficiency Comparison.....61

SUMÁRIO

1 INTRODUÇÃO 14
2 ESTRUTURAÇÃO TEÓRICA E REVISÃO DA LITERATURA 18
2.1 Aprendizado de máquina profundo e Tarefas de Computação Visual 18
2.1.1 Tarefas de Computação Visual 18
2.1.2 Modelos Clássicos de aprendizado profundo para Classificação de Imagens 19
2.2 Vision Transformer (ViT) vs. CNNs 20
2.2.1 Processo de Tokenização da imagem 21
2.2.2 Mecanismo de Auto Atenção (*Multi Head Self-Attention*) 21
2.2.3 Rede Feed-Forward (FFN) 22
2.2.4 Ativações não-lineares e Normalizações 24
2.2.5 Variações e evoluções do Modelo de Vision Transformer 25
2.3 Treinamento de Modelos de Aprendizado de Máquina por Aprendizado Supervisionado 25
2.3.1 Treinamento Convencional de modelos profundos 25
2.3.2 Treinamento com Destilação de Conhecimento 26
2.3.3 Destilação de Conhecimento em modelos DeiT (*Knowledge Distillation*) 27
2.3.4 *Fine-Tuning* 28
2.4 Impacto da dimensão dos modelos em dispositivos de borda e Escalonamento Arquitetural 28
2.5 Quantização: Descrição e desafios em modelos ViT 29
2.5.1 Classificação das quantizações 30
2.5.2 Distribuição das funções de ativação e normalizações 31
2.6 Arranjos Sistólicos 32
2.7 O acelerador *Gemmini* e System-on-Chip 33
2.7.1 Arquitetura e componentes 34
2.7.2 Configurabilidade 36
2.7.3 Interface com o acelerador - RoCC Gemmini ISA 36
2.7.4 *Tiling* de multiplicações de matriz 38
3 TRABALHOS RELACIONADOS 39
3.1 Opções de Aceleradores ViT 40
3.2 Plataformas de *Deployment* 41
3.3 Análise Comparativa 42
4 IMPLEMENTAÇÃO HARDWARE-SOFTWARE 44
4.1 Modelos Reduzidos e Treinamento de Vision Transformers 44
4.1.1 Modelos MiniViT/DeiT Propostos 45
4.1.2 Implementação do código C 46
4.1.3 Implementação das Ativações e Normalizações não-lineares em CPU 47
4.2 Esquema de treinamento para ViT 49
4.3 Esquema de Quantização Proposta 50
5 METODOLOGIA EXPERIMENTAL 53
5.1 Simuladores para execução dos experimentos 53
5.2 Avaliação do desempenho do Acelerador em FPGA 53
5.3 Implementação Física (Fluxo ASIC) 55
6 RESULTADOS EXPERIMENTAIS E DISCUSSÃO 57
6.1 Performance de Classificação e *Footprint* de Memória 57
6.1.1 Acurácia das redes 57
6.1.2 Requisitos de memória para execução dos modelos 58
6.1.3 Avaliação das diferentes redes desenvolvidas 59

6.2 Aceleração do Hardware 59
6.3 Eficiência do Hardware 61
6.3.1 Resultados do fluxo ASIC para TSMC 65nm 61
6.3.2 Comparação com dispositivos comerciais 62
7 CONCLUSÕES 63
REFERÊNCIAS 65

1 INTRODUÇÃO

Atualmente, tarefas de computação visual têm se tornado altamente requisitadas em praticamente todos os campos do conhecimento, auxiliando setores como o de inovação, com novos dispositivos de Internet-das-Coisas (IoT), bem como o reconhecimento facial presente nos *smartphones*, se estendendo até carros autônomos e equipamentos utilizados na medicina. Essa ampla presença em diversos setores se dá graças à facilitação da execução de modelos de inteligência artificial (AI), especialmente os modelos de redes neurais profundas (DNNs), ocasionada pelo advento de dispositivos que permitem execução eficiente de operações que processam uma grande quantidade de dados simultaneamente, permitindo que essas redes atinjam alta precisão em tarefas que necessitam de um amplo volume de informação para serem treinadas. Os primeiros resultados promissores foram publicados em (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), onde duas unidades de processamento gráfico (GPUs) GTX 580, foram utilizados para treinar uma rede neural convolucional (CNN) profunda para identificar uma entre 1000 classes no conjunto de dados (*dataset*) Imagenet (DENG et al., 2009).

A área de visão computacional, por muito tempo dominada por Redes Neurais Convolucionais (CNNs), testemunhou o surgimento de um novo paradigma inspirado na arquitetura Transformer, proposta por (DOSOVITSKIY et al., 2021). Os Vision Transformers (ViTs) adaptaram com sucesso o mecanismo de auto-atenção (MHSA) para tarefas visuais, tratando patches de imagem como tokens em uma sequência. Diferentemente das CNNs, que possuem fortes vieses indutivos (como localidade e equivariância à translação) que as tornam eficientes em dados limitados, os ViTs aprendem as relações espaciais quase que exclusivamente a partir dos dados. Essa característica faz com que seu desempenho seja extraordinariamente escalável: embora possam apresentar performance inferior em datasets de tamanho moderado, quando pré-treinados em coleções de dados massivas (na ordem de centenas de milhões de imagens), os ViTs demonstram uma capacidade de generalização que lhes permite superar os resultados de estado da arte das arquiteturas convolucionais. Contudo, é precisamente essa escalabilidade e o alto desempenho que trazem consigo um custo computacional elevado, o que se mostra como um desafio para implementações em dispositivos de borda, cuja capacidade de processamento e disponibilidade energética é inferior.

Nesse âmbito, a implementação de ViTs em sistemas de borda é limitada pelos seguintes fatores: a necessidade de uma execução em tempo real (ou seja, com baixa

latência), definida pelos requisitos da aplicação; a obrigatoriedade de um consumo energético significativamente baixo, devido às limitações de baterias nestes dispositivos; a impossibilidade da aplicação de modelos massivos, o que exige a minimização da quantidade de acessos à memória secundária, ou a transferência de dados da memória principal para a memória do coprocessador, no caso dos GPUs e aceleradores; e finalmente, alta precisão para realização da tarefa de computação visual.

Dispositivos como GPUs possuem ampla capacidade de execução de múltiplos dados simultaneamente, devido às suas características intrinsicamente vetoriais, permitindo um rápido processamento matricial que soluciona o problema da latência, porém causando um aumento proibitivo no consumo energético. O elevado custo energético do uso de GPUs catalisou a pesquisa e o desenvolvimento de Arquiteturas de Domínio Específico (DSAs), onde aceleradores de hardware são acoplados a um núcleo de processador de propósito geral, para acelerar cargas de trabalho específicas. Este paradigma de co-design hardware/software possibilita a criação de Systems-on-Chip (SoCs) com níveis de eficiência e desempenho inatingíveis por CPUs ou GPUs genéricas, especialmente em ambientes com restrições de energia, como em dispositivos de borda.

Baseado em processadores RISC-V, ou seja, processadores que possuem um padrão de conjunto de instruções (ISA) livre e aberto, o acelerador Gemmini, proposto pela Universidade de Berkeley (GENC et al., 2021), surge como uma solução proeminente, oferecendo um gerador de aceleradores de redes neurais de código aberto e full-stack. Integrado ao framework Chipyard (AMID et al., 2020), o Gemmini provê um array sistólico parametrizável, uma biblioteca de software e a infraestrutura para prototipagem de SoCs completos. O uso de matrizes sistólicas oferece uma plataforma unificada e centrada no fluxo de dados. Como as matrizes sistólicas se destacam na exploração da reutilização local de dados e na minimização do acesso à memória global por meio de uma grade de Elementos de Processamento (PEs) interconectados, elas são especialmente adequadas para os pesados requisitos de tiling de modelos baseados em convolução e em atenção. Ao mapear ViTs nessa arquitetura, fornecemos uma base de hardware flexível que mantém alto rendimento para transformers sem sacrificar a capacidade de executar fluxos de trabalho diversos e multimodais em um mesmo chip.

Adicionalmente, para viabilizar a implementação do ViT em dispositivos de borda com poucos recursos, adotamos um esquema de quantização robusto e eficiente para promover uma melhor utilização dos recursos, substituindo operações de ponto flutuante por cálculos simples com inteiros de 8 bits. No entanto, a quantização das ativações é particu-

larmente desafiadora para os modelos de Transformer, devido às distribuições estatísticas únicas de suas características internas. Ao contrário das distribuições do tipo gaussiano tipicamente encontradas em CNNs, os valores de ativação após as funções softmax e GELU são altamente irregulares, reduzindo consideravelmente a precisão do modelo (YUAN et al., 2021). Esse contexto demandou uma implementação de quantização que leve em conta essa característica das ativações.

Além disso, apesar da prevalência do uso de pesos pré-treinados no aprendizado profundo, existe uma lacuna na literatura quanto ao desenvolvimento de um fluxo de trabalho dedicado para treinar modelos a partir do zero, bem como o desenvolvimento de modelos miniaturizados de Vision Transformers, visando reduzir a quantidade de acessos à memória. Esse processo de treino é particularmente desafiador para os Vision Transformers (ViTs), que carecem dos vieses indutivos inerentes às CNN, como a invariância de translação, e, consequentemente, “não generalizam bem quando treinados com quantidades insuficientes de dados”. Em última análise, embora os ViTs possam ser ajustados para tarefas posteriores, os modelos treinados do zero em conjuntos de dados restritos frequentemente têm dificuldade em igualar o desempenho de seus equivalentes pré-treinados devido à falta de diversidade de dados.

Para enfrentar os desafios da implantação de Vision Transformers (ViTs) em dispositivos de borda, este trabalho propõe um fluxo abrangente de co-projeto de hardware e software. Essa metodologia inclui um fluxo de treinamento especializado, quantização por tensor adaptada à execução de ViTs, chegando até a implementação e avaliação no acelerador de redes neurais Gemmini. O treinamento é feito utilizando técnicas de destilação de conhecimento (Knowledge Distillation) e *fine-tuning*, utilizando modelos pré-treinados mais complexos (e portanto, com capacidade de detecção de padrões mais acurada) para "ensinar"modelos reduzidos de Vision Transformer.

Embora o Gemmini tenha sido originalmente projetado para inferência de Redes Neurais Convolucionais (CNN), implementamos o primeiro modelo ViT *bare-metal* para o acelerador, permitindo, com nossa estrutura, o mapeamento eficiente e a avaliação rápida de modelos baseados em ViT. Validamos nossa abordagem por meio de uma rigorosa avaliação de hardware, medindo *throughput*, latência, por meio de prototipagem em FPGA e simulação funcional, e consumo de energia, bem como estimativas de consumo de potência e área de silício sintetizadas no nó de tecnologia TSMC de 65 nm, demonstrando com sucesso a possibilidade de executar modelos menores com pouca perda em comparação com GPUs em dispositivos de borda quantizados. Para poder acomodar as

grandes tarefas de classificação em um dispositivo de borda, também propusemos versões menores de modelos bem conhecidos, como ViT (DOSOVITSKIY et al., 2021) e DeiT (TOUVRON et al., 2021). Ao abordar a atual escassez de implementações de ViT em arquiteturas de matriz sistólica, este trabalho estabelece uma linha de base para pesquisas futuras em aceleração de transformers.

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta a estruturação teórica e a revisão da literatura, abordando os conceitos de aprendizado de máquina para computação visual, a comparação estrutural entre arquiteturas Vision Transformer (ViT) e CNNs, os desafios de quantização em distribuições não-lineares e o funcionamento de arranjos sistólicos com foco no acelerador Gemmini. Em sequência, o Capítulo 3 discute os trabalhos relacionados presentes na literatura, categorizando diferentes opções de aceleradores para ViTs e plataformas de *deployment*, para então estabelecer uma análise comparativa com as contribuições deste estudo.

O Capítulo 4 descreve a implementação de Hardware e de Software adotada para a realização do trabalho. O texto engloba a proposição dos modelos miniaturizados, suas implementações em C, exportação de pesos e seus esquemas de treinamento e a metodologia de quantização mista. Posteriormente, o Capítulo 5 descreve a metodologia experimental utilizada. O texto detalha a infraestrutura de validação em FPGA, as ferramentas de simulação utilizadas e as ferramentas para a implementação física ASIC.

O Capítulo 6 expõe os resultados experimentais e as devidas discussões. Avalia-se a performance preditiva, o *footprint* de memória, a aceleração e a eficiência energética do hardware projetado sob a tecnologia TSMC 65nm, contrastando-as com dispositivos comerciais disponíveis na borda. Finalmente, Capítulo 7 sintetiza as conclusões gerais do projeto.

2 ESTRUTURAÇÃO TEÓRICA E REVISÃO DA LITERATURA

Este capítulo estabelece a fundamentação teórica para a compreensão das escolhas arquiteturais e metodológicas adotadas ao longo deste trabalho. Para viabilizar o co-design entre hardware e software e a execução eficiente de modelos de inteligência artificial na borda, é imprescindível detalhar as bases algorítmicas que regem essas redes e os desafios associados ao seu treinamento. Inicialmente, o problema do aprendizado de máquina profundo aplicado a tarefas de computação visual é contextualizado. Em seguida, a topologia das arquiteturas *Vision Transformer* (ViT) são descritas. Logo, discute-se sobre os paradigmas de treinamento supervisionado e técnicas avançadas essenciais para a adaptação de modelos ViT compactos. A seção aborda também técnicas de escalonamento de modelos e quantização. Por fim, os arranjos sistólicos e a arquitetura do acelerador gemmini são detalhadas.

2.1 Aprendizado de máquina profundo e Tarefas de Computação Visual

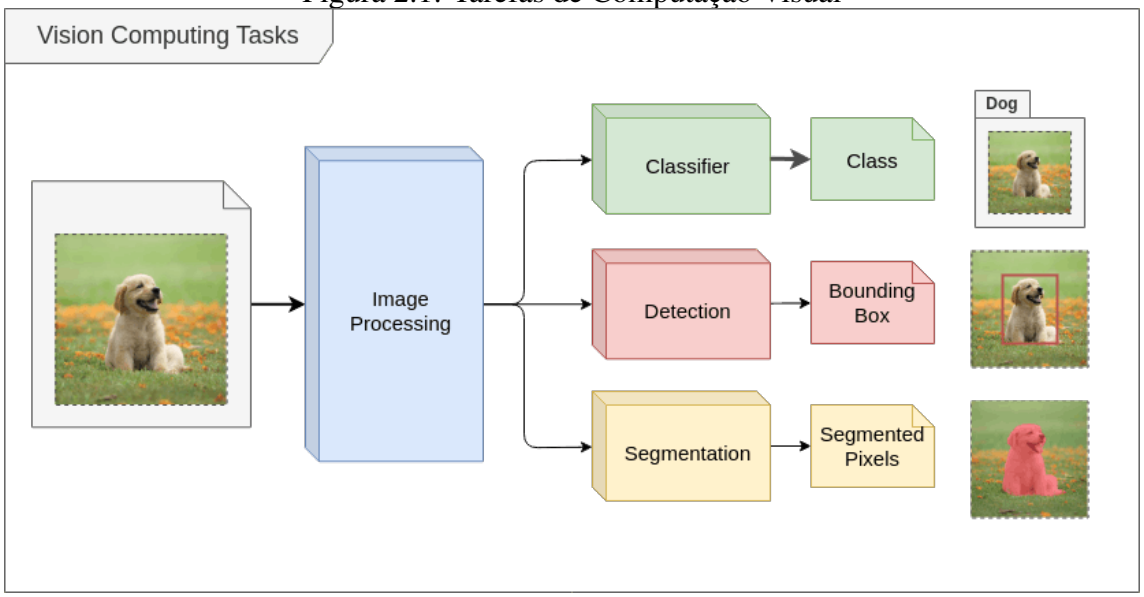
O processamento de imagens digitais representa um dos principais domínios de aplicação do aprendizado de máquina profundo. Para compreender a motivação por trás das arquiteturas modernas baseadas em atenção, é necessário estabelecer os conceitos fundamentais que regem a área. Esta seção introduz inicialmente as principais tarefas da computação visual e apresenta uma revisão dos modelos clássicos fundamentados em Redes Neurais Convolucionais (CNNs).

2.1.1 Tarefas de Computação Visual

O campo da Computação Visual foca na abstração de informações semânticas a partir de imagens, dividindo-se fundamentalmente em três níveis de granularidade. A Classificação atua de forma global, atribuindo um ou mais rótulos à cena inteira (por exemplo, categorizar uma imagem entre gato ou cachorro). A Detecção de Objetos avança ao localizar instâncias específicas através de caixas delimitadoras (*bounding boxes*). Por fim, a Segmentação atua no nível do pixel, delineando os contornos exatos de cada elemento na imagem. Essas tarefas estão exemplificadas na Figura 2.1. Este trabalho concentra-se exclusivamente na Classificação de Imagens, de forma que as demais tarefas

se encontram fora do escopo abordado.

Figura 2.1: Tarefas de Computação Visual



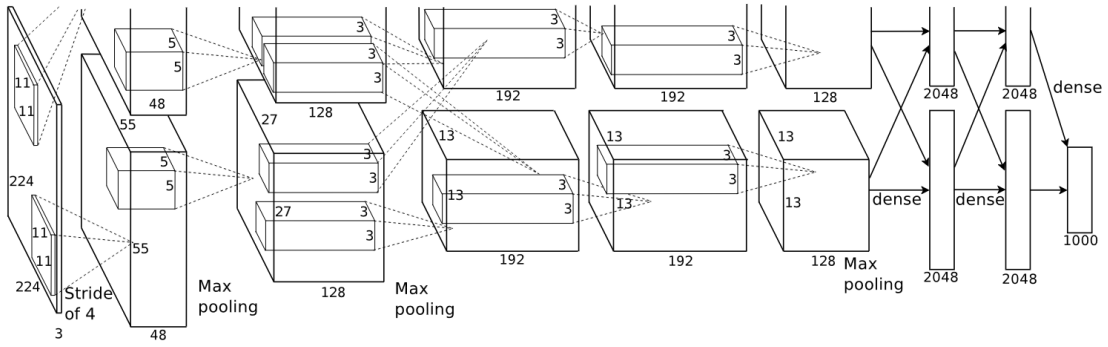
2.1.2 Modelos Clássicos de aprendizado profundo para Classificação de Imagens

Antes da introdução das arquiteturas ViT, as Redes Neurais Convolucionais (CNNs) consolidaram-se como o estado da arte para tarefas de visão computacional. A eficácia dessas redes reside na sua capacidade intrínseca de lidar com a topologia de dados em formato de grade (como os pixels de uma imagem), utilizando vieses indutivos fortes para aprender representações espaciais de maneira eficiente.

O grande ponto de virada na história da computação visual moderna ocorreu em 2012, com a introdução da rede AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Treinada no conjunto de dados ImageNet (DENG et al., 2009), um banco de dados em larga escala contendo milhões de imagens rotuladas, a AlexNet obteve uma redução drástica nas taxas de erro.

Estruturalmente, assim como descrito na Figura 2.2, modelos convolucionais clássicos operam processando a matriz de entrada através de sucessivos blocos de filtros convolucionais e operações de redução de dimensionalidade espacial (*pooling*). As camadas iniciais da rede capturam padrões de baixo nível, como bordas, cores e texturas. Conforme a informação flui pela rede, as camadas mais profundas combinam esses padrões para reconhecer partes de objetos e estruturas semânticas complexas. Após essa fase de extração hierárquica, o mapa de características resultante é achatado (*flattened*) e repas-

Figura 2.2: Descrição por camadas da rede convolucional *alexnet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012)

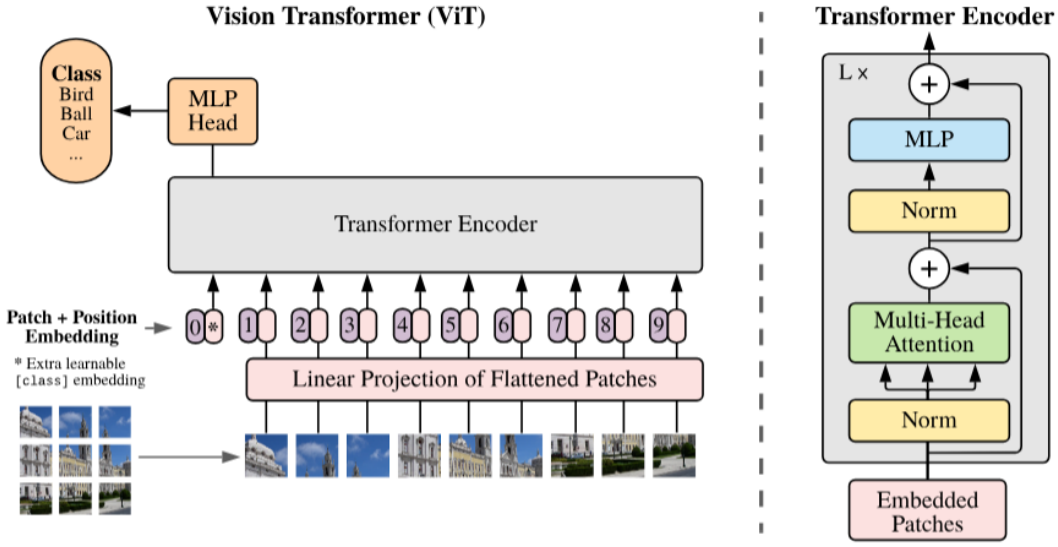


sado para camadas densas e totalmente conectadas (*fully connected layers*), que realizam o mapeamento final para as probabilidades de cada classe.

2.2 Vision Transformer (ViT) vs. CNNs

Enquanto as Redes Neurais Convolucionais (CNNs) dominaram a visão computacional por anos, o **Vision Transformer (ViT)** introduziu uma abordagem alternativa que demonstrou grande sucesso (DOSOVITSKIY et al., 2021). As CNNs possuem vieses indutivos fortes, como a localidade (filtros operam em vizinhanças de pixels) e a equivariância à translação (a identidade de um objeto não muda com sua posição), que são incorporados em cada camada. Embora esses vieses sejam úteis em cenários com poucos dados, o ViT mostra que eles não são estritamente necessários. A arquitetura ViT trata uma imagem como uma sequência de "patches"(pequenos recortes), que são então processados por um Transformer padrão, similar a como palavras são tratadas em tarefas de NLP (VASWANI et al., 2023). Embora o ViT tenha menos vieses indutivos específicos para imagens, ele compensa essa lacuna ao ser pré-treinado em conjuntos de dados massivos (14M a 300M de imagens), como o Imagenet-21K (RIDNIK et al., 2021). Com dados em escala suficiente, o ViT aprende os padrões espaciais relevantes do zero e supera o desempenho das CNNs de ponta em diversas tarefas de classificação, exigindo, ao mesmo tempo, menos recursos computacionais para o treinamento. O diagrama na Figura 2.3 descreve a topologia de um Vision Transformer.

Figura 2.3: Topologia Vision Transformer (DOSOVITSKIY et al., 2021)



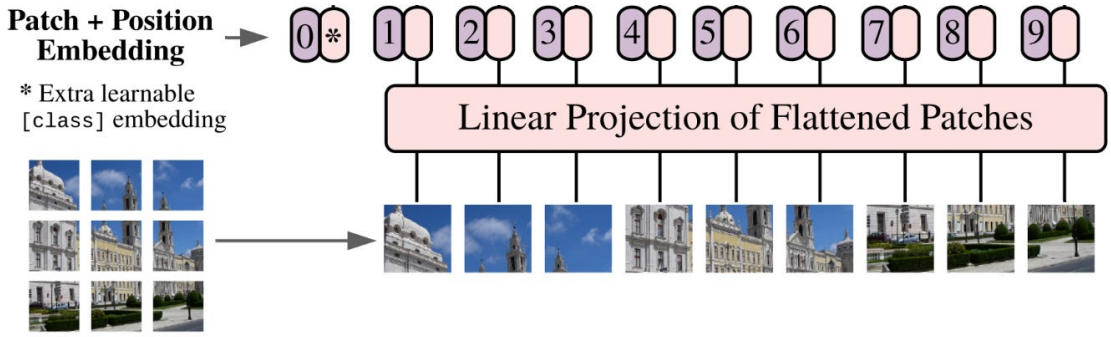
2.2.1 Processo de Tokenização da imagem

A transformação de uma imagem em uma sequência de dados processáveis inicia-se pelo processo de patch embedding, no qual a imagem de entrada é dividida em pequenas sub-regiões de tamanho fixo denominadas patches. Cada um desses patches é achatado (flattened) e submetido a uma projeção linear para um espaço latente de dimensão específica, convertendo-os efetivamente em tokens vetoriais. Devido à ausência de vieses indutivos espaciais inerentes às redes convolucionais, os ViTs tratam esses tokens de forma agnóstica à sua ordem original. Para que o modelo possa interpretar a estrutura geométrica da imagem, é imperativo injetar informações sobre a posição relativa ou absoluta de cada token por meio de positional encodings. Essas codificações posicionais, que podem ser baseadas em funções senoidais ou aprendidas, possuem a mesma dimensão dos embeddings de tokens e são somadas a eles antes da entrada no codificador, garantindo que o mecanismo de autoatenção consiga correlacionar as características visuais respeitando a topologia original da imagem.

2.2.2 Mecanismo de Auto Atenção (*Multi Head Self-Attention*)

A arquitetura **Transformer**, apresentada originalmente para tarefas de tradução automática, dispensou completamente o uso de recorrência e convoluções, baseando-se unicamente em mecanismos de atenção. O componente central é o mecanismo de auto-

Figura 2.4: Processo de separação em *patches* e incorporação de posição (*Position Embedding*). (DOSOVITSKIY et al., 2021)



atenção (self-attention), ilustrado na Figura 2.5 e descrita matematicamente na Equação 2.1, que permite ao modelo ponderar a importância de diferentes partes da sequência de entrada para gerar uma representação de cada elemento. Essencialmente, a auto-atenção relaciona todas as posições da sequência entre si, calculando "pesos de atenção"que determinam o quanto cada elemento deve "prestar atenção"a todos os outros. Isso é feito mapeando uma consulta (query) e um conjunto de pares chave-valor (key-value) para uma saída, que é uma soma ponderada dos valores. A grande vantagem desse mecanismo é sua capacidade de capturar dependências de longo alcance com um número constante de operações, ao contrário das redes recorrentes (RNNs), que exigem um caminho sequencial que dificulta o aprendizado de tais relações. Além disso, a ausência de recorrência permite uma paralelização massiva durante o treinamento, tornando o processo significativamente mais rápido. O uso de **Multi-Head Attention** (ou *Multi Head Self-Attention*) permite que o modelo foque em diferentes subespaços de representação em paralelo, enriquecendo sua capacidade de capturar padrões complexos (VASWANI et al., 2023).

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

2.2.3 Rede Feed-Forward (FFN)

A Rede Feed-Forward (FFN) no Vision Transformer é uma subcamada fundamental presente em cada bloco do codificador, posicionada logo após o mecanismo de autoatenção (Multi-Head Self-Attention). O principal objetivo da FFN é introduzir não-linearidade no modelo, permitindo que a rede processe e transforme as características extraídas pelo mecanismo de atenção em representações mais complexas e úteis para a

Figura 2.5: Topologia do mecanismo de Multi Head Self-Attention, (VASWANI et al., 2023)

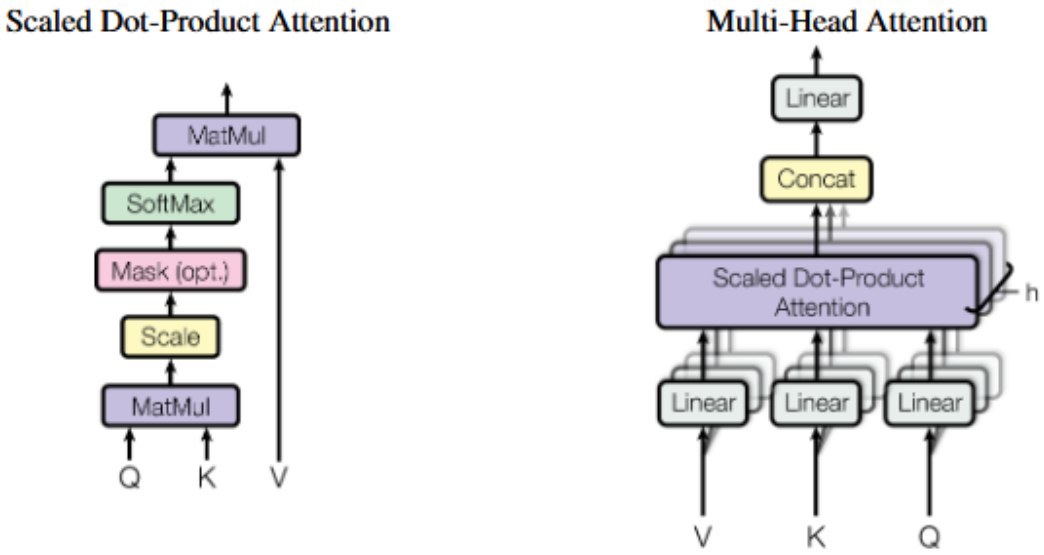
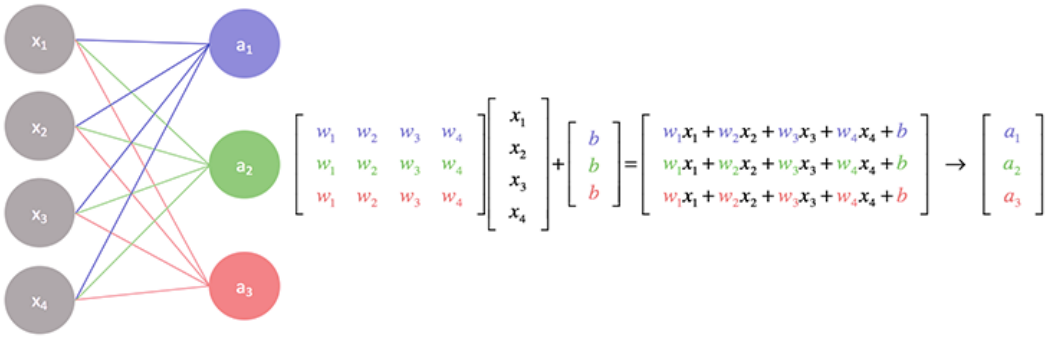


Figura 2.6: Representação de uma FFN, ou rede Fully-Connected como uma operação de GEMV (JORDAN, 2017)



tarefa final, como a classificação de imagens. Esse módulo consiste em duas camadas totalmente conectadas (Fully Connected Network - FCN) com uma função de ativação não linear entre elas, geralmente uma Gaussian Error Linear Unit (GELU), que proporciona uma não-linearidade mais suave em comparação à ReLU, favorecendo a convergência de modelos de larga escala. Por ser formada por FCNs, esse módulo pode ser reduzido a operações GEMV, representado na Figura 2.6, de forma que apresenta alto *throughput* no arranjo sistólico.

2.2.4 Ativações não-lineares e Normalizações

Ademais, as funções de ativação e normalização desempenham um papel crítico na capacidade de representação dos Vision Transformers, mas introduzem desafios significativos para a implementação em hardware de baixa precisão devido à natureza de suas distribuições estatísticas.

Softmax: A função Softmax, presente no mecanismo de autoatenção, é responsável por converter os valores de similaridade em uma distribuição de probabilidade, garantindo que a soma dos pesos de atenção seja igual a um. A função softmax está descrita na Equação 2.2.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.2}$$

Layer Normalization: Esta operação é aplicada antes das subcamadas de atenção e FFN para estabilizar a dinâmica de treinamento e normalizar a escala das ativações ao longo da dimensão do modelo, de acordo com a figura 2.3. A Equação 2.3 descreve a operação de Layernorm, onde μ e σ^2 são a média e a variância dos elementos de x, respectivamente, enquanto γ e β são parâmetros aprendíveis de escala e deslocamento.

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta \tag{2.3}$$

Para fins de otimização de hardware, os pesos γ e β podem ser fixados em $\gamma = 1$ e $\beta = 0$, porém necessitando retreinamento da rede. Esse processo resulta na Equação 2.4.

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.4}$$

GELU (Gaussian Error Linear Unit): A GELU é a principal função de ativação utilizada nas redes feed-forward (FFN), substituindo a ReLU (Rectified Linear Unit) por oferecer uma transição mais suave e melhor desempenho em modelos de larga escala, porém introduzindo um custo computacional elevado. Essa função é representada pela Equação 2.5, onde "erf"é a função erro, descrita na Equação 2.6.

$$\text{GELU}(x) = \frac{1}{2}x \left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{2.5}$$

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{2.6}$$

2.2.5 Variações e evoluções do Modelo de Vision Transformer

Com o avanço do estado da arte, novos modelos baseados em ViT foram propostos para mitigar limitações intrínsecas da arquitetura original ou para oferecer alternativas a aplicações específicas. Nesse âmbito, surgiram modelos como o DeiT (*Data Efficient Image Transformer*) (TOUVRON et al., 2021) e o TinyViT (WU et al., 2022), cujo objetivo é otimizar o treinamento com conjuntos reduzidos de dados. Essa estratégia permite alcançar acurácias mais altas mesmo sem acesso a *datasets* de larga escala, sem exigir grandes alterações na arquitetura em comparação com o modelo ViT base (DOSOVITSKIY et al., 2021). Por outro lado, redes como o MobileViT (MEHTA; RASTEGARI, 2022) propõem famílias de modelos desenvolvidos para dispositivos de borda, utilizando conceitos similares aos da arquitetura MobileNet (HOWARD et al., 2017). Essa adaptação, no entanto, se dá através de mudanças significativas na topologia da rede, tornando seu desenvolvimento em aceleradores de borda não convencionais um desafio. Por fim, modelos como o *Swin Transformer* (*Shifted Window Transformer*) (LIU et al., 2021) propõem alterações no mecanismo de *patching* da imagem. Ao utilizar quadros (*patches*) que se sobrepõem, a arquitetura aprimora a identificação de padrões locais, de modo a atingir acurácias mais elevadas.

2.3 Treinamento de Modelos de Aprendizado de Máquina por Aprendizado Supervisionado

O treinamento de redes neurais profundas para tarefas de visão computacional baseia-se no paradigma do aprendizado supervisionado, em que um modelo ajusta iterativamente seus parâmetros internos a partir de um conjunto de dados previamente rotulados. Esta seção detalha as metodologias de otimização estrutural adotadas para garantir a convergência e a precisão dessas arquiteturas.

2.3.1 Treinamento Convencional de modelos profundos

No contexto do aprendizado de máquina profundo, o treinamento de Redes Neurais Convolucionais (CNNs) e Perceptrons Multicamadas (MLPs) convencionais fundamenta-se no algoritmo de retropropagação do erro (backpropagation), um processo iterativo es-

truturado em duas etapas principais que visa à minimização de uma função de custo. Inicialmente, na etapa de alimentação direta (feedforward), os dados de entrada propagam-se pelas camadas da rede, sofrendo transformações lineares e ativações não lineares, até a geração de uma previsão final na camada de saída. Em seguida, calcula-se o erro da rede ao mensurar a discrepância entre essa previsão e o rótulo real através da função de perda; essa penalidade é então retropropagada em direção às camadas iniciais por meio da regra da cadeia do cálculo diferencial, determinando o gradiente do erro em relação a cada peso e viés (bias) do modelo. Finalmente, um otimizador baseado no gradiente descendente utiliza esses valores para atualizar os parâmetros da arquitetura na direção oposta ao gradiente, reduzindo progressivamente o erro e refinando a capacidade de generalização e extração de características da rede ao longo das épocas de treinamento.

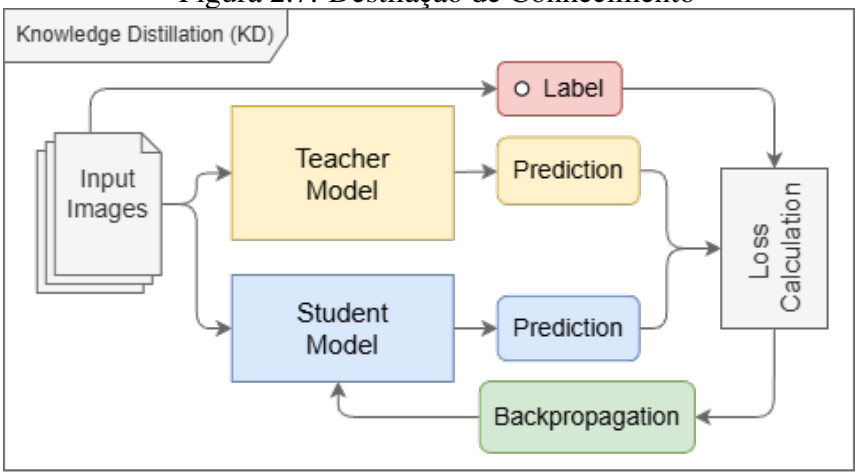
2.3.2 Treinamento com Destilação de Conhecimento

A destilação de Conhecimento, ou *Knowledge Distillation*, é uma técnica amplamente utilizada em aprendizado de máquina para transferir as capacidades de um modelo maior (professor) para um modelo menor (estudante ou *student*), atingindo uma acurácia final significativamente superior ao processo clássico de treino. Nessa técnica de transferência de aprendizado, um modelo de alta capacidade (professor) orienta o treinamento de um modelo mais leve e eficiente (aluno). Em vez de focar apenas no acerto do rótulo correto, o processo ensina o aluno a imitar a forma como o professor generaliza e compreende a similaridade dos dados, utilizando "alvos suaves"(soft targets), as probabilidades relativas de todas as classes ajustadas por um fator de "temperatura"para revelar relações secundárias ocultas. Em resumo, essa técnica ensina a rede não apenas para acertar a classificação, mas também garantir que a distribuição de probabilidades das classes (ou até mesmo de camadas internas) seja semelhante ao modelo professor.

Para mitigar a limitação causada pela ausência de viéses indutivos em bases menores, a destilação de conhecimento (Knowledge Distillation) é amplamente empregada en modelos ViT. Essa técnica permite, não apenas auxiliar na identificação de padrões, mas também no aprendizado de vieses indutivos, que são facilmente obtidos em CNNs e apresentam um desafio para modelos baseados em Vision Transformers, sendo assim, uma das técnicas mais importantes para o treinamento desses modelos. O diferencial dessa estratégia reside no fato de que, ao contrário do processo de treinamento clássico em que o valor utilizado para atualizar os pesos é calculado apenas pelo erro ou acerto da

classificação final (para modelos de classificação), a função de perda depende também da distribuição de probabilidades da última camada (antes do classificador). Dessa forma, o modelo é capaz de apresentar um comportamento similar ao modelo professor, permitindo aprender características como vieses indutivos e identificação de padrões. Essa estratégia é ilustrada na Figura 2.7.

Figura 2.7: Destilação de Conhecimento



2.3.3 Destilação de Conhecimento em modelos DeiT (*Knowledge Distillation*)

Os modelos DeiT (TOUVRON et al., 2021), propõem abordagens distintas para treinar redes Vision Transformers (ViTs) de forma eficiente, mitigando a dependência histórica de bases de dados colossais. O modelo é treinado utilizando apenas a base ImageNet de tamanho padrão, compensando a falta de dados com técnicas severas de aumento de dados (como Rand-Augment, Mixup e Cutmix) e implementando uma destilação baseada em um token de destilação, que interage com as camadas de autoatenção para reproduzir a decisão ditada por um modelo professor, preferencialmente convolucional.

A mecânica central empregada pelas arquiteturas da família DeiT (TOUVRON et al., 2021) inova ao introduzir um vetor que pode ser treinado à sequência de entrada, denominado *token* de destilação. De forma análoga ao funcionamento do *token* de classe, o *token* de destilação interage globalmente com os demais fragmentos da imagem (os *patch tokens*) iterativamente através das camadas de *self-attention*. No entanto, seus objetivos de otimização divergem: enquanto o `CLS` é atualizado para minimizar o erro em relação ao rótulo verdadeiro da imagem, o objetivo exclusivo do *token* de destilação é reproduzir a saída preditiva do modelo professor.

Durante o processo de *backpropagation*, a rede otimiza uma função de custo conjunta. Adotando a estratégia de destilação rígida (*hard-label distillation*), a perda total do modelo é calculada como uma combinação linear entre a perda de entropia cruzada tradicional e a perda de destilação, expressa matematicamente pela Equação 2.7:

$$\mathcal{L}_{total} = \frac{1}{2}\mathcal{L}_{CE}(\psi(Z_{cls}), y) + \frac{1}{2}\mathcal{L}_{CE}(\psi(Z * dist), y_t) \tag{2.7}$$

onde $\mathcal{L}_{CE}$ representa a função de entropia cruzada, ψ denota a função de ativação *softmax*, Z_{cls} e Z_{dist} são as saídas numéricas brutas (*logits*) dos respectivos *tokens*, y é o rótulo verdadeiro da amostra e y_t é a predição rígida (a classe de maior probabilidade) ditada pelo modelo professor. Essa abordagem arquitetural força o *Transformer* estudante a herdar ativamente a topologia de decisão e os vieses indutivos extraídos pela rede CNN, acelerando a convergência espacial e maximizando a precisão em bases de dados menores.

2.3.4 *Fine-Tuning*

O ajuste fino (*fine-tuning*) é uma técnica de transferência de aprendizado na qual um modelo de rede neural, previamente treinado em um conjunto de dados de larga escala, tem seus parâmetros ajustados para solucionar uma nova tarefa específica. Em vez de inicializar os pesos da arquitetura de forma aleatória e realizar o treinamento desde o início, o modelo mantém a capacidade de extração de características visuais genéricas (como bordas, formas e texturas) já consolidadas na etapa de pré-treinamento. Durante o procedimento, é prática comum congelar os pesos das camadas iniciais da rede e retreinar exclusivamente as camadas finais responsáveis pela classificação, ou aplicar uma taxa de aprendizado reduzida em toda a estrutura. Essa abordagem diminui o custo computacional, acelera o tempo de convergência e mitiga o risco de *overfitting* quando o novo conjunto de dados possui um volume restrito de amostras.

2.4 Impacto da dimensão dos modelos em dispositivos de borda e Escalonamento Arquitetural

O dimensionamento das redes neurais possui um impacto direto no design de aceleradores de hardware, uma vez que parâmetros como a largura, profundidade e resolução ditam as restrições de área, consumo de potência e largura de banda de memória (PAT-

TERSON; HENNESSY, 2020). Em arquiteturas dedicadas, o tamanho das matrizes de pesos e dos mapas de características determina a quantidade de acessos à memória on-chip e a taxa de utilização das unidades de processamento. Quando a dimensionalidade do modelo excede a capacidade de armazenamento, o sistema sofre com penalidades de latência devido ao tráfego constante de dados. Portanto, parametrizar e otimizar a arquitetura da rede para caber no orçamento de recursos do hardware é essencial para projetos eficientes e viáveis em silício.

Historicamente, arquiteturas de Redes Neurais Convolucionais (CNNs) foram pioneiras na exploração desse compromisso computacional. A arquitetura MobileNet (HOWARD et al., 2017) introduziu multiplicadores de largura para afinar uniformemente o número de canais em cada camada, reduzindo de forma quadrática o custo computacional e o número de parâmetros para viabilizar a inferência rápida em sistemas embarcados restritos. Expandindo essa visão, a família EfficientNet (TAN; LE, 2020) comprovou empiricamente que o escalonamento isolado de apenas uma dimensão da rede (como alargar excessivamente os canais) gera retornos decrescentes de acurácia. Para resolver isso, a EfficientNet estabeleceu um método de escalonamento composto, que utiliza um coeficiente constante para balancear simultaneamente a largura, a profundidade e a resolução da rede, garantindo a maximização do desempenho num cenário de recursos restritos.

Para modelos de Vision Transformer, a literatura demonstra que é viável escalar as dimensões de um modelo (*scaling down*), reduzindo significativamente sua complexidade computacional sem incorrer em perdas severas de acurácia (ZHAI et al., 2022). Ao equilibrar estrategicamente atributos estruturais, como a largura, a profundidade e a resolução da rede, o impacto no desempenho preditivo final pode ser consideravelmente mitigado. Contudo, essa miniaturização exige que a nova arquitetura reduzida seja devidamente retreinada ou submetida a um processo de ajuste fino (*fine-tuning*), assegurando que os pesos se adaptem à topologia restrita e preservem a capacidade de generalização visual do modelo original.

2.5 Quantização: Descrição e desafios em modelos ViT

A execução da inferência do Vision Transformer (ViT), bem como de demais modelos de DNNs, utilizando pesos de ponto flutuante de 32 bits (FP32) costuma representar um uso subótimo dos recursos de hardware (NAGEL et al., 2021). A maioria das redes neurais apresenta uma redundância significativa, o que lhes permite funcionar com uma

perda de precisão negligenciável, mesmo quando se utiliza apenas uma fração da largura de bits original. A quantização resolve isso mapeando valores de alta precisão para uma representação de menor precisão. Para cada valor, um fator de escala (X_{scale}) e um ponto zero (Q_{Zero}) são calculados e armazenados junto com os valores inteiros quantizados, seguindo a relação descrita na equação 2.8 (NAGEL et al., 2021):

$$X_{int8} = \frac{X_{FP32} - Q_{Zero}}{Q_{scale}} \tag{2.8}$$

2.5.1 Classificação das quantizações

A classificação das técnicas de quantização abrange a precisão numérica, a granularidade das escalas, a simetria do ponto zero e o momento de aplicação (PTQ ou QAT), definindo o equilíbrio fundamental entre a fidelidade do modelo e a eficiência energética do acelerador de hardware.

Precisão: A quantização pode ser usada para reduzir um número em representação de ponto flutuante para qualquer quantidade de bits, tanto em ponto flutuante de menor precisão, como uma representação de inteiro. O uso da precisão de 8 bits inteiro (INT8) representa um equilíbrio: não requer técnicas de compensação excessivamente robustas e alcança quase a mesma precisão que o FP32, ao mesmo tempo em que reduz os requisitos de armazenamento em aproximadamente 75%. Outras larguras de bits apresentam desafios distintos:

- INT16: Apresenta perda de precisão insignificante, mas não oferece compressão significativa do modelo nem aumento de velocidade.
- INT4: Reduz drasticamente o tamanho do modelo, mas frequentemente requer estratégias de mitigação complexas, como precisão mista ou granularidade extremamente fina.
- Larguras de bits não padronizadas: Valores entre larguras padrão de potência de dois quebram o alinhamento de memória, aumentando a complexidade do caminho de dados e potencialmente estendendo o caminho crítico do acelerador, que é frequentemente determinado por instruções de movimentação de memória para registro, no caso do acelerador Gemmini.

Granularidade: Além disso, a quantização pode ser aplicada em vários níveis de granularidade. Embora as abordagens de granularidade grossa sejam mais fáceis de

implementar, elas podem prejudicar mais a precisão do modelo (XIAO et al., 2024). Por outro lado, a quantização de granularidade fina oferece maior precisão, mas aumenta significativamente o número de escalas FP32 que precisam ser armazenadas, exigindo um controle de fluxo de dados mais complexo no hardware.

Posição do Zero: A fim de reduzir a sobrecarga computacional e potencializar a vazão de dados no acelerador, pode-se estabelecer o valor nulo para o ponto zero ($Q_{Zero} = 0$). Tal estratégia define a quantização simétrica, permitindo que as operações aritméticas sejam executadas de forma mais célere e econômica ao simplificar o processamento em arquiteturas sistólicas.

Relação com o treinamento: No contexto da otimização de redes neurais para implantação em hardware, a quantização pode ser realizada de duas formas: a Quantização Pós-Treinamento (PTQ) e o Treinamento Consciente de Quantização (QAT) (GHOLAMI et al., 2021). A diferença entre as duas reside no momento de aplicação e na capacidade de adaptação do modelo aos erros de precisão. A técnica de PTQ é executada após o término do treinamento, incidindo sobre um modelo com pesos já congelados e utilizando um conjunto reduzido de dados de calibração para definir os intervalos de quantização, o que a torna uma solução de baixo custo computacional e rápida implementação. Em contrapartida, o QAT insere a simulação dos efeitos da quantização diretamente no ciclo de treinamento ou ajuste fino (*fine-tuning*), permitindo que os pesos da rede se adaptem dinamicamente para compensar a perda de informação numérica.

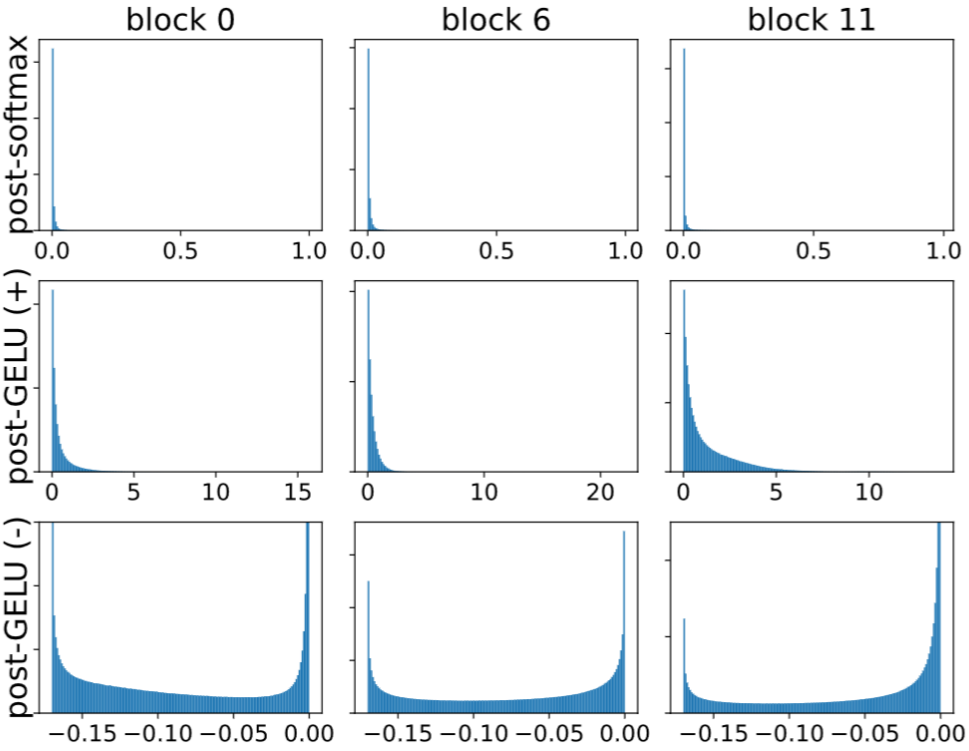
2.5.2 Distribuição das funções de ativação e normalizações

A Figura 2.8 ilustra as distribuições estatísticas das ativações em diferentes blocos de um ViT, destacando por que a quantização tradicional é desafiadora nessas arquiteturas:

- Post-softmax (Primeira linha): Mostra valores altamente desbalanceados no intervalo $[0, 1]$, onde a grande maioria está próxima de zero, mas alguns valores próximos a um são cruciais para o mecanismo de atenção.
- Post-GELU (Duas linhas inferiores): Demonstra uma distribuição assimétrica após a função de ativação GELU. Enquanto os valores positivos (+) se estendem por uma faixa ampla (até 15 ou 20), os valores negativos (-) estão confinados a um intervalo muito pequeno, aproximadamente entre $-0,17$ e 0.

Essas distribuições não-Gaussianas e assimétricas tornam difícil o uso de um único

Figura 2.8: Distribuição da normalização Softmax e da função de ativação GELU (YUAN et al., 2021)



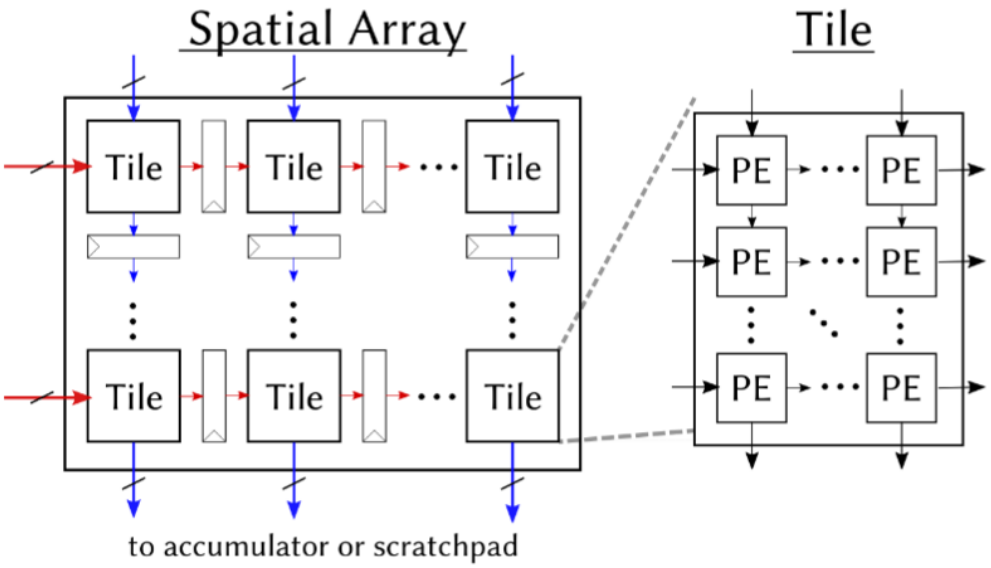
fator de escala na quantização uniforme, pois focar nos valores grandes gera erros nos pequenos e vice-versa (YUAN et al., 2021).

2.6 Arranjos Sistólicos

A arquitetura sistólica foi concebida para superar o gargalo de I/O (entrada/saída) em computações intensivas, conhecidas como compute-bound. Em arquiteturas convencionais, o desempenho é frequentemente limitado pela largura de banda da memória, pois cada operação exige a busca de um ou dois operandos da memória. Arranjos sistólicos resolvem esse problema ao maximizar a reutilização de dados. A ideia central é que um dado, uma vez lido da memória, flui ritmicamente através de uma malha de elementos de processamento (PEs) simples e interconectados, sendo utilizado em múltiplas computações antes de ser descartado ou retornar à memória. Esse fluxo de dados, análogo à circulação sanguínea, permite que o sistema atinja uma alta taxa de processamento sem aumentar a demanda por largura de banda da memória (KUNG, 1982). A estrutura regular e modular dos arranjos sistólicos simplifica o design, a implementação e a expansão,

tornando-os ideais para operações matriciais como multiplicação e convolução, que são fundamentais em redes neurais. A Figura 2.9 descreve a topologia do arranjo sistólico.

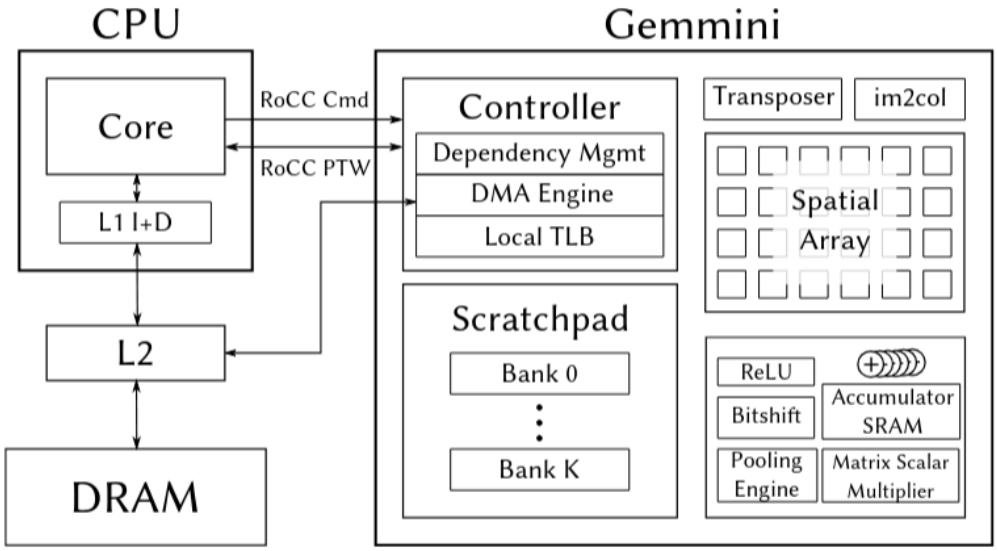
Figura 2.9: Microarquitetura do Arranjo Sistólico Gemmini (GENC et al., 2021)



2.7 O acelerador *Gemmini* e System-on-Chip

O acelerador de redes neurais profundas (DNNs) *Gemmini* é um co-processador desenvolvido sobre o ecossistema *Chipyard* (AMID et al., 2020), desenvolvido na Universidade da Califórnia Berkeley (UC Berkeley), pelo grupo de pesquisa UC Berkeley Architecture Research, tendo como base a linguagem de descrição de hardware (HDL) alto nível *Chisel*, baseada em *Scala*. A partir desse sistema, é possível gerar diferentes versões do acelerador Gemmini, incluindo a possibilidade de integração com processadores RISC-V e demais periféricos e componentes, compondo um System-on-Chip (SoC), o qual está descrito na Figura 2.10. Esse SoC é composto por um processador Rocket RISC-V, uma memória principal L2 e uma interface para DRAM, além do acelerador Gemmini. A interconexão intra-chip é feita pelo protocolo TileLink, além do uso da interface RoCC (Rocket Chip Co-processor), que será descrita em detalhes na seção 2.7.3, para repasse de instruções do processador para o acelerador.

Figura 2.10: System-on-Chip Gemmini (GENC et al., 2021)



2.7.1 Arquitetura e componentes

A estrutura do Gemmini é composta por blocos modulares que gerenciam de forma eficiente a computação intensiva e o movimento de dados.

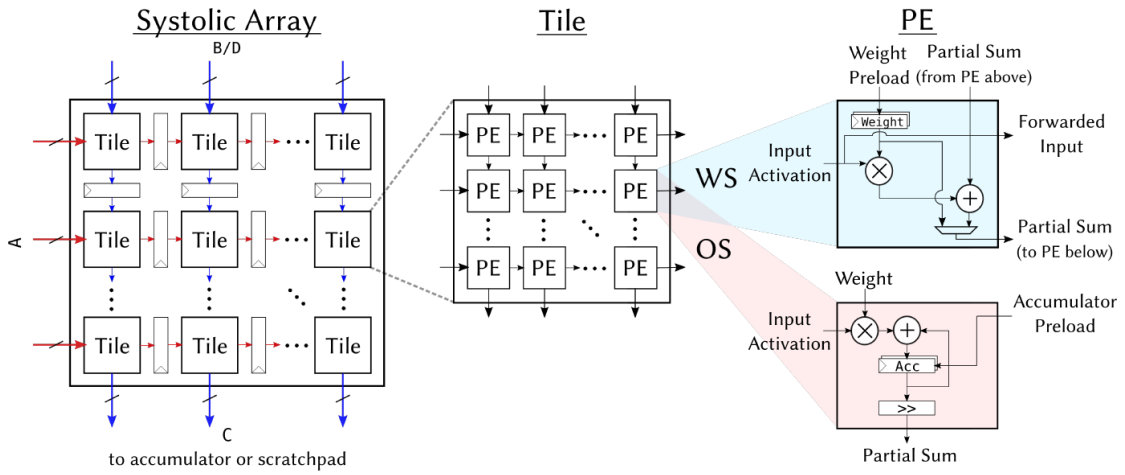
Controlador (Controller): Atua como a parte de controle do coprocessador, sendo responsável por decodificar as instruções customizadas enviadas pelo processador *host* através da interface RoCC e, com base nas instruções recebidas, estimular os sinais de controle corretos para a parte operacional. Ele coordena a sincronização entre as subtarefas de carga, execução e armazenamento, garantindo que as dependências de dados sejam respeitadas durante o processamento paralelo.

Arranjo Sistólico (Systolic Array): É o núcleo computacional do sistema, formado por uma malha bidimensional de elementos de processamento (PEs) interconectados. Cada arranjo sistólico é considerado como um *Tile* do acelerador, que pode ou não conter múltiplos tiles.

Elementos de Processamento (PEs): Representam as unidades aritméticas fundamentais dentro do arranjo. Cada PE é equipado com lógica para operações de multiplicação e acumulação (MAC), operando sobre dados quantizados, como INT8, e suportando diferentes estratégias de fluxo de dados (dataflow), como weight-stationary (WS) ou output-stationary(OS), para otimizar a localidade temporal e espacial. A hierarquia e a microarquitetura dos PEs está descrita na Figura 2.11.

Scratchpad: Consiste em uma memória SRAM local de alta velocidade utilizada

Figura 2.11: Microarquitetura dos elementos de processamento (PE) dentro da hierarquia de 2 níveis, composta por *Tiles* de *PEs*



para o armazenamento temporário de operandos (pesos e ativações), utilizados como *inputs*. Sua função principal é servir como um buffer de baixa latência que alimenta o arranjo sistólico, mitigando os gargalos de largura de banda inerentes à memória principal do sistema.

Acumulador (Accumulator): É uma unidade de memória especializada, geralmente com maior largura de bits (como INT32), dedicada a armazenar somas parciais de multiplicações matriciais e resultados das operações. O acumulador permite a realização de somas acumulativas e operações de bias-addition antes que os dados sejam submetidos ao pós-processamento e eventual requantização para o formato de saída.

Unidade de DMA (Direct Memory Access): Gerencia a transferência assíncrona de grandes blocos de dados entre a memória principal (DRAM) e as memórias locais (Scratchpad e Acumulador). Gemmini possui duas unidades de DMA, uma para processo de leitura e outra para processos de escrita na memória principal. O uso de um DMA dedicado permite o mascaramento da latência de memória, possibilitando que o acelerador carregue novos dados enquanto processa cálculos aritméticos simultaneamente.

Blocos de Pós-processamento e Ativação: Incluem circuitos especializados para a aplicação de funções de ativação, como ReLU, e operações de escalonamento (scaling). Estes blocos são fundamentais para o suporte à quantização, transformando os resultados de alta precisão do acumulador de volta para o formato de 8 bits (INT8) utilizado na inferência eficiente de modelos como os Vision Transformers.

2.7.2 Configurabilidade

A flexibilidade do Gemmini é implementada através de um sistema de parâmetros em Chisel que permite a geração de instâncias de hardware customizadas para cargas de trabalho específicas, como os Vision Transformers.

Tipos de Dados e Precisão Numérica: O acelerador permite a definição independente da largura de bits e do formato (inteiro ou ponto flutuante) para as entradas, saídas e acumuladores. Esta característica possibilita a implementação de esquemas de quantização otimizados, como o INT8, equilibrando a redução do tamanho do modelo com a preservação da acurácia necessária para a aplicação.

Dimensões do Arranjo Sistólico: A geometria da malha computacional é parametrizável em termos de número de linhas e colunas de elementos de processamento (PEs). Essa escalabilidade permite que o designer ajuste o nível de paralelismo aritmético conforme os recursos físicos disponíveis, adequando o acelerador desde pequenos dispositivos de borda até sistemas mais robustos.

Hierarquia de Memória Local: As capacidades do Scratchpad e do Acumulador são ajustáveis, permitindo o aumento dessas memórias para evitar gargalos em modelos com grandes volumes de parâmetros ou ativações. No processamento de ViTs, por exemplo, a ampliação dessas dimensões é fundamental para acomodar as demandas específicas das camadas de atenção, de modo que todo o modelo seja carregado no scratchpad, evitando assim um acesso sub-ótimo à memória.

Unidades Funcionais Customizadas: O Gemmini permite a inclusão modular de blocos de pós-processamento, como diferentes funções de ativação e normalizações. Esta modularidade assegura que apenas o hardware necessário para a aplicação alvo seja instanciado, evitando o desperdício de área em silício com funcionalidades não utilizadas.

2.7.3 Interface com o acelerador - RoCC Gemmini ISA

O conjunto de instruções (Instruction Set Architecture - ISA) do Gemmini é implementado como uma extensão customizada do RISC-V, utilizando a interface RoCC para delegar operações matriciais complexas ao acelerador. As instruções são projetadas para gerenciar o fluxo de dados bidimensionais e configurar o comportamento aritmético do arranjo sistólico. As instruções são principalmente divididas em: Instruções de Movimentação de Dados, Instruções de Configuração e Instruções de Controle e Especi-

alizadas. São elas:

- **`mvin` (Move Data From Main Memory to Scratchpad):** Esta instrução é responsável por transferir matrizes bidimensionais da memória principal (DRAM) para a memória local do acelerador (scratchpad ou acumulador).
- **`mvout` (Move Data from Scratchpad to DRAM):** Realiza a operação inversa, movendo os resultados processados do acumulador para a memória principal ou cache L2. Além da transferência de dados, o mvout possui suporte para a execução de *max-pooling* diretamente durante o processo de escrita, permitindo que parâmetros como tamanho da janela, stride e preenchimento (padding) sejam aplicados conforme os dados são armazenados no formato NHWC.
- **`config_ex` (Configure Execute Pipeline):** Define os parâmetros fundamentais da execução aritmética no arranjo sistólico. Através desta instrução, o programador seleciona o fluxo de dados (dataflow) entre as opções weight-stationary ou output-stationary, habilita funções de ativação como ReLU e configura transposições automáticas para as matrizes de entrada. Ela também estabelece o fator de escalonamento necessário para converter os resultados de alta precisão do acumulador (accType) de volta para o formato de entrada (inputType).
- **`config_mvin` e `config_mvout`:** Estas instruções configuram os parâmetros de acesso à memória para as operações de carga e armazenamento, respectivamente. Elas definem os strides (passos de memória) tanto na memória principal quanto na memória privada do acelerador. No caso do `config_mvin`, é possível ainda configurar um fator de escala para multiplicar os dados durante o carregamento, enquanto o `config_mvout` define as dimensões da imagem resultantes de operações de pooling.
- **`flush`:** Esta instrução gerencia a consistência da memória ao realizar o flush do TLB (Translation Lookaside Buffer) do acelerador. Diferente das instruções aritméticas, o flush possui execução imediata assim que recebido, sendo responsabilidade do programador garantir a correta sincronização através de *fences* para evitar conflitos com instruções de memória que ainda estejam em fila.

2.7.4 *Tiling* de multiplicações de matriz

O acelerador Gemmini só é capaz de processar multiplicações de matrizes de tamanho $DIM \times DIM$, onde DIM é a dimensão do arranjo sistólico, em PEs. Para permitir multiplicações de matrizes de dimensões maiores, é necessário aplicar *tiling* nas matrizes, ou seja, subdividi-las. Em vez de processar linhas e colunas inteiras de uma matriz, o que pode exceder a capacidade do arranjo sistólico, a computação é dividida em submatrizes menores chamadas tiles ou blocos.

Para dar suporte a matrizes maiores, Gemmini possui também um conjunto de instruções CISC (Complex Instruction Set Computer), também conhecidas como instruções de laço (Loop Instructions); são extensões de alto nível projetadas para realizar operações de multiplicação matricial e convolução em volumes de dados que excedem as dimensões físicas do arranjo sistólico ($DIM \times DIM$).

Devido à alta complexidade e ao grande número de operandos, estas instruções não cabem em uma única instrução customizada RISC-V. Elas são implementadas como sequências de múltiplas instruções de hardware que devem ser chamadas consecutivamente, estando disponíveis para uso através da biblioteca gemmini.h no ambiente de software do acelerador.

Loop de Multiplicação de Matrizes (`gemmini_loop_ws`): Esta operação executa o cálculo matricial $A \times B + D = C$ para matrizes de grandes dimensões, utilizando o fluxo de dados weight-stationary (WS). Enquanto as matrizes A e B devem obrigatoriamente utilizar o tipo de dado de entrada (inputType), as matrizes D e C oferecem flexibilidade, podendo operar tanto em inputType quanto no formato de maior precisão do acumulador (accType).

3 TRABALHOS RELACIONADOS

Diversos trabalhos buscaram otimizar a execução de Vision Transformers com foco em dispositivos de baixo consumo para plataformas edge. Para isso, foi desenvolvido em várias frentes de inovação maneiras de reduzir (1) o tamanho dos modelos em memória, (2) a latência e a taxa de inferências por segundo, (3) e a potência consumida, sem reduzir significativamente a acurácia dos modelos.

Nesse contexto, um dos métodos de atingir esse objetivo sem necessitar de grandes alterações no hardware é por meio da quantização, entretanto, essa técnica exige atenção quando se trata de Vision Transformers. Assim como foi demonstrado em (YUAN et al., 2021) e mencionado na Seção 2.5.2, modelos de Vision Transformer utilizam normalizações e cálculos de probabilidade que possuem, como saída, distribuições não-gaussianas, que dificultam a operação de cálculos de quantização de forma precisa. Trabalhos como (YUAN et al., 2021), (HUANG et al., 2023), (LI et al., 2022), (NAG et al., 2023)propuseram diferentes soluções para lidar com essas distribuições em aceleradores dedicados. Do outro lado, outros trabalhos propuseram outras soluções para quantização focando em GPUs, que entretanto servem como base de comparação, como a implementação do NVIDIA FasterTransformer (NVIDIA, 2023) e (LI; GU, 2023).

Ademais, a plataforma de desenvolvimento escolhida também é crítica pensando na operação ótima dos Vision Transformers. Trabalhos como (NAG et al., 2023) e (YOU et al., 2025) propuseram soluções em ASIC, diferentemente dos demais que buscaram implementações em FPGAs, que permite uma maior flexibilidade sob custo de frequências mais baixas de operação e menor espaço de design.

Finalmente, o último critério a ser analisado é a arquitetura de acelerador desenvolvida. Alguns trabalhos utilizaram arranjos sistólicos, como (HUANG et al., 2023). Como mencionado na Seção 2.6, essas unidades de processamento são desenvolvidas para obter alta performance em operações de multiplicação de matrizes (GEMM). Por outro lado, demais trabalhos utilizaram unidades de processamento vetorial, desenvolvidas para obter alta eficiência em operações Matriz-Vetor (GEMV), que é o caso dos trabalhos (NAG et al., 2023) e (WANG et al., 2022).

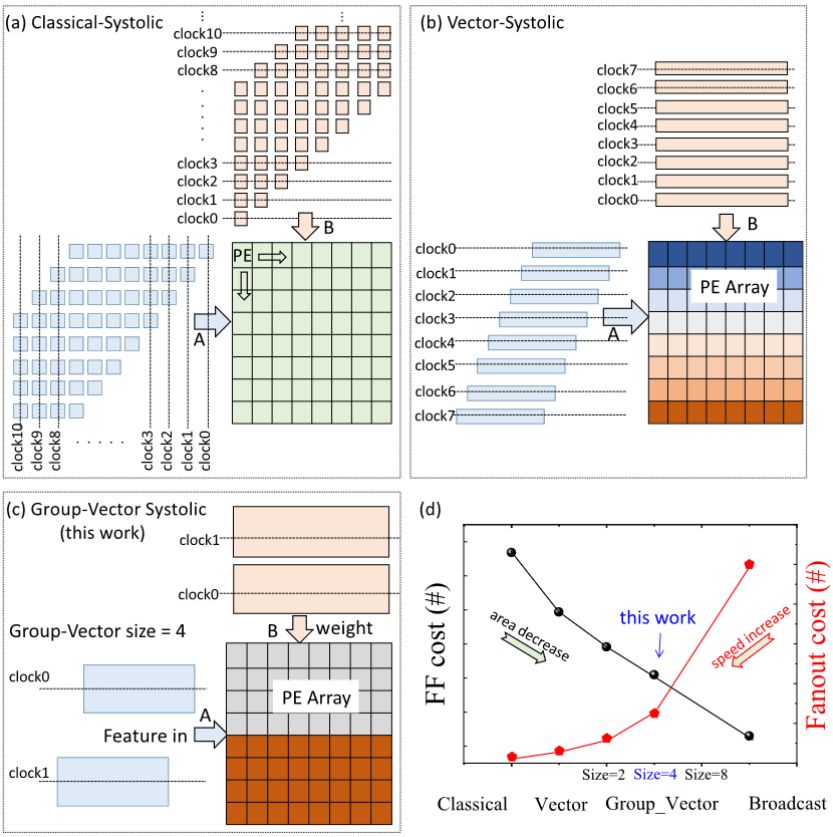
Essa seção discorre sobre os diferentes tipos de aceleradores disponíveis, a fim de justificar a escolha do arranjo sistólico.

3.1 Opções de Aceleradores ViT

Dentre as possibilidades de arquitetura de aceleradores disponíveis, temos os aceleradores baseados em arranjo sistólico, como em (HUANG et al., 2023), no qual Huang et al. utiliza uma variação de arquitetura: é utilizado um Arranjo Sistólico de Vetor de Grupo (Group Vector Systolic Array), o qual apresenta uma redução do pipeline em relação ao modelo clássico de arranjo sistólico, como é implementado na TPU da Google (JOUPPI et al., 2017). Essa configuração permite reduzir o número de flip-flops (FFs) intermediários, estabelecendo um *trade-off* entre frequência de operação e eficiência energética e área de silício. Quanto às funções especiais, como Layer Normalization, Softmax e GELU, Huang et al. utiliza unidades especiais para realizar os cálculos apenas utilizando inteiros, logo todas as etapas dos cálculos, descritos na Seção 2.2.4, são mapeados para operações de soma, multiplicação e deslocamento de bits em inteiros de 8 bits, com acumuladores de maior largura de bits para evitar *overflow*. Essa abordagem é vista também em outros trabalhos como (LI; GU, 2023) e (KIM et al., 2021). Uma comparação entre essa implementação de arranjo e a implementação clássica está descrita na Figura 3.1. O trabalho apresenta uma implementação para FPGA. Embora a arquitetura descrita no trabalho aparente apresentar elementos não sintetizáveis, o fluxo de desenvolvimento de circuitos integrados não foi levada em consideração, o que apresenta resultados subótimos para a arquitetura sistólica. Ademais, o trabalho também apresenta quantização de ativações em 8 bits, o que pode apresentar degradação nos resultados se técnicas de treinamento ciente de quantização (vide Seção 2.5.1) não forem aplicadas propriamente.

Outra possibilidade, como apresentada em (LI et al., 2022), (NAG et al., 2023), e (WANG et al., 2022), é a utilização de estruturas que se assemelham a NPUs, como a Google Coral NPU (GOOGLE, 2026). Nessa arquitetura, em vez de organizados em uma malha, como no caso do arranjo sistólico, os elementos de processamento (PEs) são organizados em um vetor 1D. Essa implementação faz sentido uma vez que o sistema é projetado para ser implantado em uma FPGA. Utilizar um arranjo sistólico em uma FPGA geralmente causa problemas de roteamento dado as características de fabricação do FPGA, que será explicado em detalhes na seção 3.2. A Figura 3.2 descreve a implementação do acelerador ViTA, que possui uma estrutura de elementos de processamento organizados de forma vetorial. Do ponto de vista da quantização, esse acelerador utiliza quantização de ponto fixo em partes da execução e quantização em potência de 2 em outras partes, para evitar sobreutilização de DSPs (Digital Signal Processors) na imple-

Figura 3.1: Comparação entre as diferentes implementações de Arranjo Sistólico (HUANG et al., 2023)

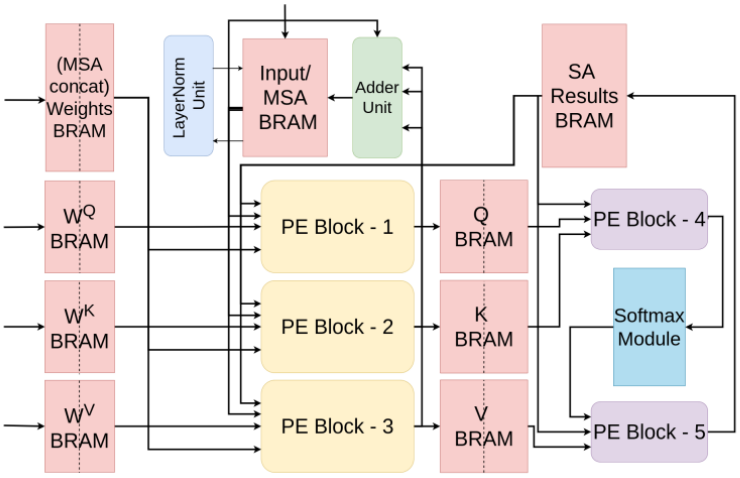


mentação em FPGA. A largura de bits utilizada é determinada a partir de uma restrição de latência, que é utilizada em uma exploração de espaço de design (Design Space Exploration - DSE) para reduzir a largura de bits até atingir uma implementação que respeita a restrição.

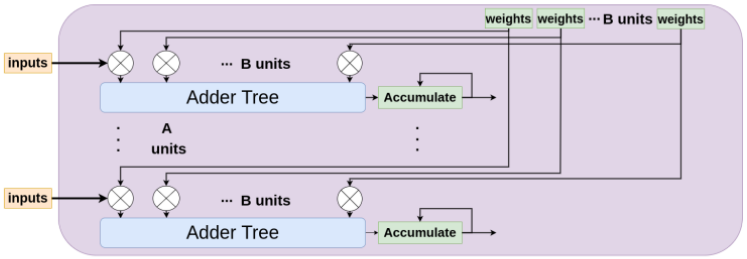
3.2 Plataformas de *Deployment*

Além da escolha da arquitetura, uma outra decisão importante no projeto de aceleradores para redes neurais é a decisão de uma plataforma que dê o suporte necessário para determinada arquitetura. Em geral, pode-se desenvolver hardwares dedicados de duas maneiras: utilizando Field Programmable Gate Arrays (FPGAs) ou partir para um processo de desenvolvimento e fabricação de um Application Specific Integrated Circuit (ASIC). A decisão de utilizar cada uma das alternativas representa um *trade-off* claro: FPGAs podem ser reprogramadas, tornando a tarefa do desenvolvimento de hardware, uma tarefa de descrição de hardware alto nível utilizando linguagens como VHDL e SystemVerilog, além de permitir tempos de desenvolvimento mais curtos. Por outro lado, o

Figura 3.2: Arquitetura proposta pelo acelerador ViTA (NAG et al., 2023)



(a) Overall Hardware Accelerator Design



(b) PE block architecture - the rows share weights. For PE Blocks - 1, 2 & 3, $A{=}k_1$ & $B{=}k_2$, and for PE Blocks 4 & 5, $A{=}k_3$ & $B{=}k_4$

desenvolvimento de um ASIC, embora exija um conhecimento técnico mais aprofundado (utilização das ferramentas de síntese e domínio dos processos físicos e elétricos), um custo de fabricação mais elevado e um maior tempo de produção, essa alternativa permite atingir frequências de operação significativamente maiores utilizando frações da potência dissipada em FPGAs, apresentando ganhos em termos de energia e latência, sob pena de não permitir a reprogramação do circuito.

3.3 Análise Comparativa

Em síntese, a literatura recente focada na otimização de Vision Transformers para dispositivos edge demonstra que a busca por eficiência envolve um complexo trade-off entre precisão, consumo de energia, área física e latência. A técnica de quantização consolida-se como a estratégia predominante para viabilizar esses modelos em hardwares com recursos limitados; contudo, o sucesso de sua aplicação exige o desenvolvimento de unidades aritméticas especializadas ou adaptações no design para lidar com as distribuições não-gaussianas (como Softmax e Layer Normalization) características dos Transfor-

mers.

Além das técnicas de compressão de dados, a literatura recente provê uma ampla diversidade de implementações estruturais voltadas para aceleração de *Vision Transformers*. Diferentes trabalhos propõem topologias de unidades de processamento e exploram múltiplas plataformas de *deployment*, disponibilizando alternativas arquiteturais para mitigar o alto custo computacional dessas redes na borda. Para situar a contribuição deste trabalho em relação ao estado da arte, a Tabela 3.1 estabelece uma análise comparativa com alguns aceleradores recentes.

A comparação fundamenta-se em quatro características primárias de projeto. Primeiramente, avalia-se a aplicação de técnicas de **Quantização** para a redução da precisão numérica do modelo. Em seguida, analisa-se a **Ciência de Ativações não-gaussianas**, um critério crítico que indica se o hardware e o fluxo de dados preveem tratamentos específicos para as distribuições irregulares e *outliers* gerados nativamente por funções não-lineares descritos na Seção 2.5.2. Por fim, avalia-se o emprego de uma **Arquitetura Sistólica** para a maximização do reúso de dados no processamento denso de matrizes (GEMM) e se o sistema foi validado e **Desenvolvido para ASIC**, buscando o estado da arte em métricas de área e eficiência energética em oposição a implementações restritas à prototipação em FPGAs.

Tabela 3.1: Comparação de características de Aceleradores para ViT recentes

Trabalho	**Quantização**	**Ciente de Ativações não-gaussianas**	**Arquitetura Sistólica**	**Desenvolvido para ASIC**
(HUANG et al., 2023)	✓	✓	✓	✗
(LI et al., 2022)	✓	✓	✗	✗
(WANG et al., 2022)	✗	✗	✗	✗
(NAG et al., 2023)	✓	✓	✗	✓
(YOU et al., 2025)	✗	✗	✗	✓
Este trabalho	✓	✓	✓	✓

4 IMPLEMENTAÇÃO HARDWARE-SOFTWARE

O desenvolvimento de aceleradores dedicados à inferência de modelos de aprendizado profundo exige uma abordagem integrada, conhecida na literatura por Hardware-Software Co-Design. Nessa abordagem, as restrições físicas do hardware influenciam diretamente a topologia e o treinamento das redes neurais, e vice-versa, ou seja, fazem-se escolhas de topologias e esquemas de quantização que beneficiem a execução eficiente em Hardware. Esse processo contou com o desenvolvimento de Vision Transformers em linguagem *Python* e de um *framework* que permite realizar as seguintes tarefas:

1. Treinar redes neurais desenvolvidas em linguagem *Python* (usando o framework *pytorch*) com o auxílio de técnicas para lidar com os problemas inerentes da implementação de modelos baseados em *Transformer*, permitindo o uso de múltiplos datasets;
2. Avaliar o resultado de acurácia das diferentes redes treinadas, permitindo variação nas dimensões dos modelos em um alto nível de abstração;
3. Realizar a quantização dos pesos gerados pelo treinamento para diferentes precisões;
4. Exportar os pesos e escalas da quantização para um arquivo *header* na linguagem C.

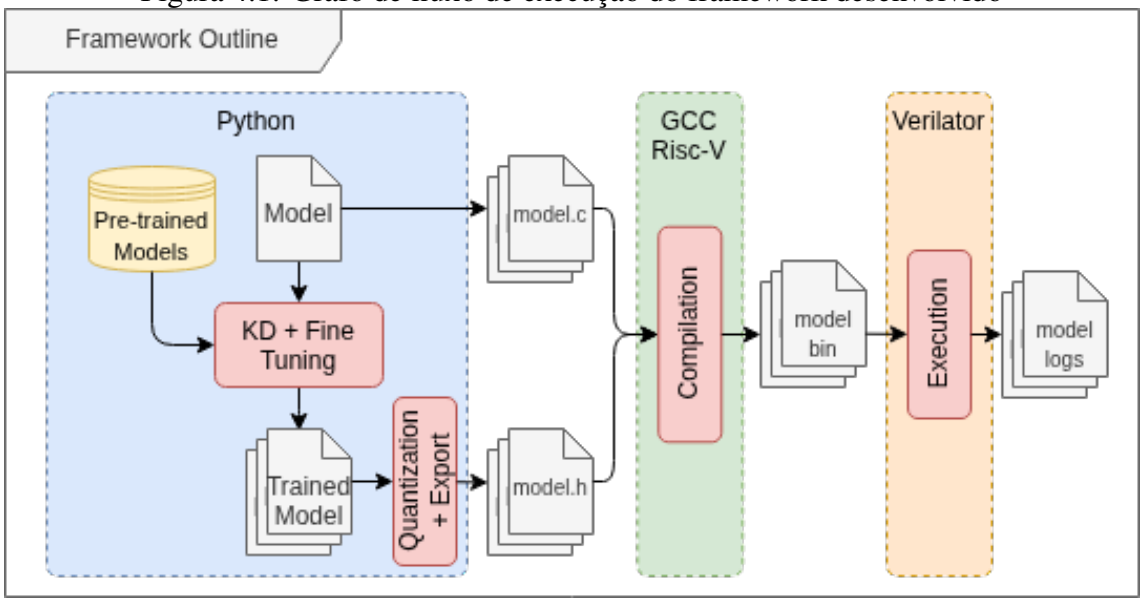
A imagem na Figura 4.1 resume como o framework desenvolvido opera desde o desenvolvimento de um modelo em pytorch até a geração de código executável para o Gemmini, bem como extração de *logs* de execução, permitindo uma rápida depuração. Vale ressaltar que o código gerado a partir da Compilação (Compilation) pode ser utilizado no acelerador em qualquer plataforma, seja ela simuladores Spike ou Verilator, FPGA ou até mesmo ASIC.

Este capítulo detalha a metodologia de *Hardware-Software Co-Design* adotada para viabilizar a execução eficiente de arquiteturas *Vision Transformers* (ViTs) em dispositivos de borda (*edge devices*).

4.1 Modelos Reduzidos e Treinamento de Vision Transformers

O padrão mais utilizado no desenvolvimento de aplicações ViT se dá por meio do reaproveitamento de pesos pré-treinados em *datasets* de larga escala, uma vez que

Figura 4.1: Grafo de fluxo de execução do framework desenvolvido



só a partir destes é possível permitir que o mecanismo de Auto-Atenção aprenda padrões visuais ao nível do estado da arte. Entretanto, os modelos disponíveis *online* de livre-acesso, treinados para *datasets* como o ImageNet21k (RIDNIK et al., 2021) e o JFT-300M possuem um número elevado de parâmetros e consequentemente de FLOPs (Floating Point Operations), o que traz dificuldades intrínsecas ao desenvolvimento em plataformas que necessitam de baixa latência e reduzido uso de memória. Buscando solucionar esse problema, esse trabalho se propõe a fazer uma busca em um espaço de design (DSE, ou *Design Space Exploration*), procurando atingir uma performance reduzida em relação ao estado-da-arte, porém utilizando apenas uma fração dos recursos de computação. Essa abordagem, apesar de proporcionar uma redução significativa em termos de energia consumida e latência de inferência, traz consigo a necessidade de um esquema de treinamento sólido, considerando as dificuldades em treinar o mecanismo de auto atenção (DOSOVITSKIY et al., 2021).

4.1.1 Modelos MiniViT/DeiT Propostos

Este trabalho implementa versões miniaturizadas e customizadas das arquiteturas Vision Transformer (ViT) (DOSOVITSKIY et al., 2021) e Data-efficient Image Transformer (DeiT) (TOUVRON et al., 2021) para viabilizar a inferência sob as restritas limitações de memória e latência de dispositivos de borda, utilizando práticas de Escalonamento Arquitetural como descritos na Seção 2.4. As versões miniaturizadas possuem dimensões

que foram avaliadas empiricamente conforme os resultados obtidos no treinamento. Os parâmetros utilizados para os modelos MiniViT estão descritos na Tabela 4.1, já os modelos miniaturizados MiniDeiT estão descritos na Tabela 4.2. Quanto à arquitetura, o modelo DeiT, proposto em (TOUVRON et al., 2021), diferencia-se estruturalmente do ViT original pela presença de um token de destilação, que potencializa o aprendizado por meio de transferência de conhecimento, contornando o desafio de treinar esses modelos do zero em bases de dados pequenas (vide Seção 2.3.2).

Tabela 4.1: Dimensões estruturais dos modelos propostos com base na arquitetura MiniViT

Parâmetro	**MiniViT Tiny**	**MiniViT Small**	**MiniViT Base**	**MiniViT Large**
Dimensão Oculta (D)	96	144	192	384
Heads de Atenção (H)	2	3	3	6
Camadas de Transformer (L)	4	6	8	12
Parâmetros Treináveis	490K	1.57M	3.64M	21.45M

Tabela 4.2: Dimensões estruturais dos modelos propostos com base na arquitetura MiniDeiT

Parâmetro	**MiniDeiT Tiny**	**MiniDeiT Small**	**MiniDeiT Base**	**MiniDeiT Large**
Dimensão Oculta (D)	64	144	192	384
Heads de Atenção (H)	2	3	3	6
Camadas de Transformer (L)	4	8	10	12
Parâmetros Treináveis	261K	2.02M	4.57M	21.45M

4.1.2 Implementação do código C

A execução dessas redes neurais no acelerador Gemmini exige um mapeamento das operações de tensores para as instruções nativas da arquitetura do acelerador. Para permitir a execução da inferência nas redes neurais avaliadas no *System-on-Chip*, a abordagem aplicada neste trabalho foi implementar réplicas das arquiteturas em linguagem C. A construção dessas réplicas baseou-se na topologia dos grafos exportados pelo PyTorch, garantindo que a sequência de operações refletisse o modelo originalmente validado em *software*.

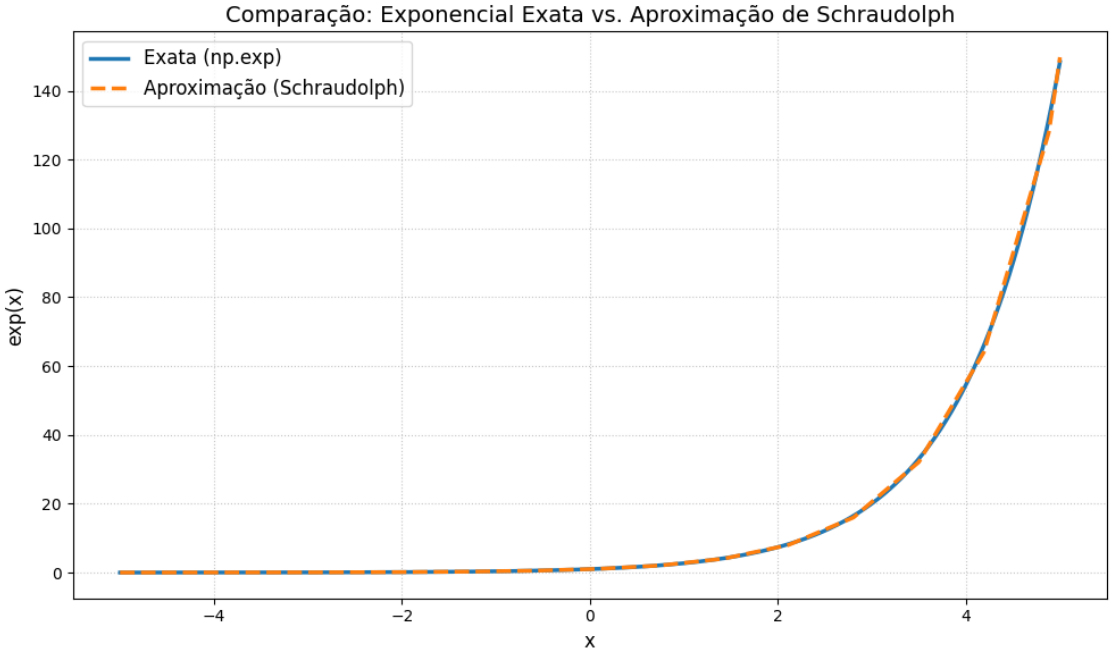
Para a codificação das arquiteturas de rede neural em linguagem C, utilizou-se o conjunto de instruções e interfaces de programação de alto nível (APIs) disponibilizadas

no repositório *gemmini-rocc-tests*, desenvolvido pela Universidade de Berkeley, à exceção das ativações e normalizações, as quais foram implementadas à parte. O processo de conversão do software consistiu no mapeamento direto das matrizes e operações matemáticas. Dessa forma, a estrutura sequencial do código em C foi construída replicando-se com exatidão a topologia do grafo gerado pelo PyTorch durante a fase de modelagem.

4.1.3 Implementação das Ativações e Normalizações não-lineares em CPU

Para a execução das operações não lineares de normalização e ativação descritas na Seção 2.5.2, foi necessária a implementação de uma aproximação para a exponencial. Nesse trabalho foi empregada a aproximação por meio do Método de Schraudolph (SCHRAUDOLPH, 1999), descrita no Algoritmo 1. Um Gráfico comparando as duas implementações é disposto na Figura 4.2.

Figura 4.2: Aproximação de Schraudolph em comparação com o cálculo exato da exponencial



As funções não-lineares para execução em CPU das funções não-lineares foram implementadas seguindo as equações descritas nessa seção. A equação 4.1 apresenta uma implementação que beneficia a estabilidade numérica ao subtrair o maior valor da linha

Algorithm 1 Aproximação Rápida da Exponencial (Método de Schraudolph)

Require: Valor de entrada $x \in \mathbb{R}$
Ensure: Aproximação $y \approx e^x$
if $x \leq -88.0$ **then**
    return 0.0 ▷ Evita underflow extremo
else if $x \geq 88.0$ **then**
    $x \leftarrow 88.0$ ▷ Evita overflow de precisão simples
end if
$c_1 \leftarrow 12102203.0$ ▷ Constante de escala $\approx 2^{23}/\ln(2)$
$c_2 \leftarrow 1064986824$ ▷ Termo de bias do expoente IEEE 754
$I \leftarrow \lfloor c_1 \cdot x + c_2 \rceil$ ▷ Cálculo no domínio dos inteiros de 32 bits
$y \leftarrow \text{bitcast_to_float}(I)$ ▷ Reinterpretação de bits (Type Punning)
return y

de cada elemento, reduzindo drasticamente o valor obtido após a exponenciação.

$$y_i = \frac{\exp(x_i - m)}{\sum_{j=1}^{C} \exp(x_j - m)} \tag{4.1}$$

As equações 4.2, 4.3 e 4.4 apresentam o cálculo passo a passo da normalização *Layer Normalization*, implementada de acordo com (BA; KIROS; HINTON, 2016). Na Equação 4.2 é calculado a média dos elementos de cada linha do tensor, então na Equação 4.3 é calculado o desvio padrão de cada linha do tensor, por fim, cada elemento tem seu valor final calculado através da Equação 4.4.

$$\mu = \frac{1}{C} \sum_{j=1}^{C} x_j \tag{4.2}$$

$$\sigma = \sqrt{\frac{1}{C} \sum_{j=1}^{C} (x_j - \mu)^2 + \epsilon} \tag{4.3}$$

$$y_j = \frac{x_j - \mu}{\sigma} \tag{4.4}$$

Por fim, a função de ativação GELU é calculada através da Equação 4.5, que aproxima-se da implementação precisa descrita por (HENDRYCKS; GIMPEL, 2023).

$$\text{GELU}(x) = 0.5x \left[1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + 0.044715x^3) \right) \right] \tag{4.5}$$

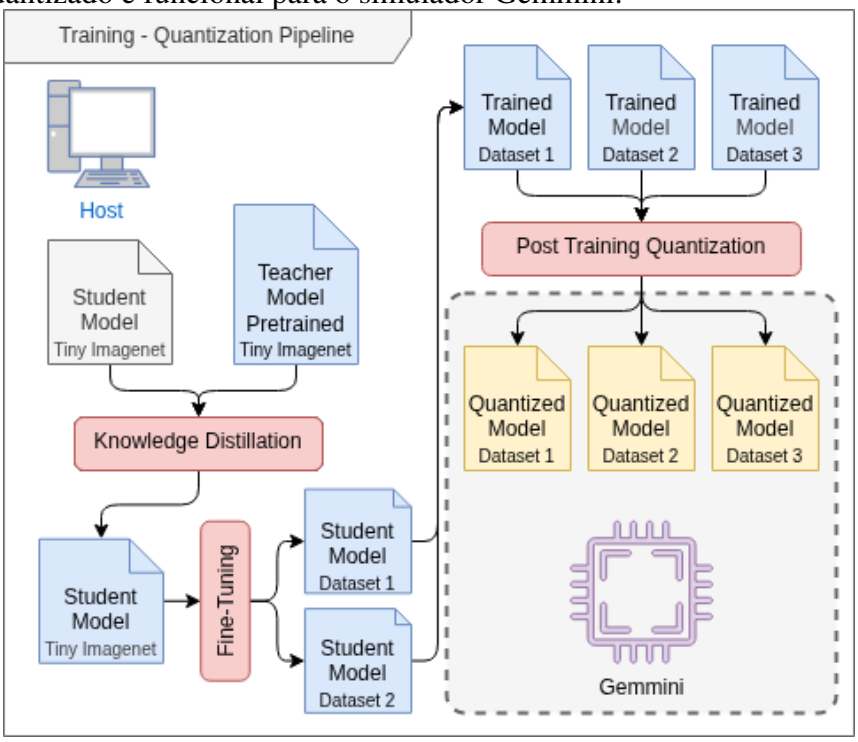
4.2 Esquema de treinamento para ViT

Modelos de Vision Transformer possuem dificuldades no aprendizado de padrões visuais, de modo que dependem de um conjuntos de dados massivos para treinamento, devido a ausência de vieses indutivos. Para treinar redes de Transformer para conjuntos de dados reduzidos, o trabalho descrito em (DOSOVITSKIY et al., 2021) implementa um esquema de treinamento incluindo transferência de conhecimento (HAO et al., 2022). Entretanto, realizar o treino de redes neurais com recursos computacionais limitados se torna um desafio para grandes conjuntos de dados como o Imagenet-21K (RIDNIK et al., 2021), necessitando muitos dias de computação em GPUs modernas para cada modelo. Buscando reduzir a complexidade dessas tarefas, modelos como DeiT (TOUVRON et al., 2021) buscam aplicar técnicas para otimizar o processo de Transferência de Conhecimento. Com base nos artigos anteriores, esse trabalho propõe um esquema de treinamento pensado em solucionar as dificuldades dos ViTs.

Com base nos fundamentos apresentados na Seção 2.3.2, a metodologia proposta neste trabalho implementa um pipeline de treinamento focado em viabilizar a implantação no hardware. Para o processo de destilação, adotou-se uma arquitetura ResNet50 (HE et al., 2015), previamente treinada no conjunto de dados ImageNet, sob o qual foi aplicado *fine-tuning* para o dataset TinyImagenet, que atuou como modelo professor. Essa rede orientou o aprendizado das arquiteturas miniaturizadas (modelos estudantes) durante as tarefas de classificação nessa mesma base. Em seguida, para adaptar o aprendizado aos demais conjuntos de dados (como CIFAR-10 e CIFAR-100), realizou-se um processo de ajuste fino (fine-tuning) congelando todos os pesos do *backbone* do Transformer e retreinando exclusivamente a camada final de classificação (classifier head). Essa estratégia, descrita na Figura 4.3 garantiu a reutilização eficiente de pesos extraídos para tarefas visuais correlatas.

Após a extração de características visuais e do processo de destilação no *dataset* primário (TinyImageNet), a adaptação do modelo para novas tarefas de inferência, como a classificação nas bases CIFAR-10 e CIFAR-100, é realizada através do ajuste fino (*fine-tuning*). O principal desafio nesta etapa é preservar o conhecimento herdado do modelo professor, evitando a degradação dos pesos, um fenômeno conhecido como esquecimento catastrófico (*catastrophic forgetting*) (LUO et al., 2025). Para contornar esse problema adotou-se a estratégia de transferência de aprendizado com congelamento estrutural (*freezing*). Neste procedimento, todas as matrizes de pesos pertencentes ao *backbone* do

Figura 4.3: Pipeline de execução do Framework, partindo de um modelo estudante e um modelo professor pré-treinado, passando por um processo de *fine-tuning*, chegando até o modelo quantizado e funcional para o simulador Gemmini.

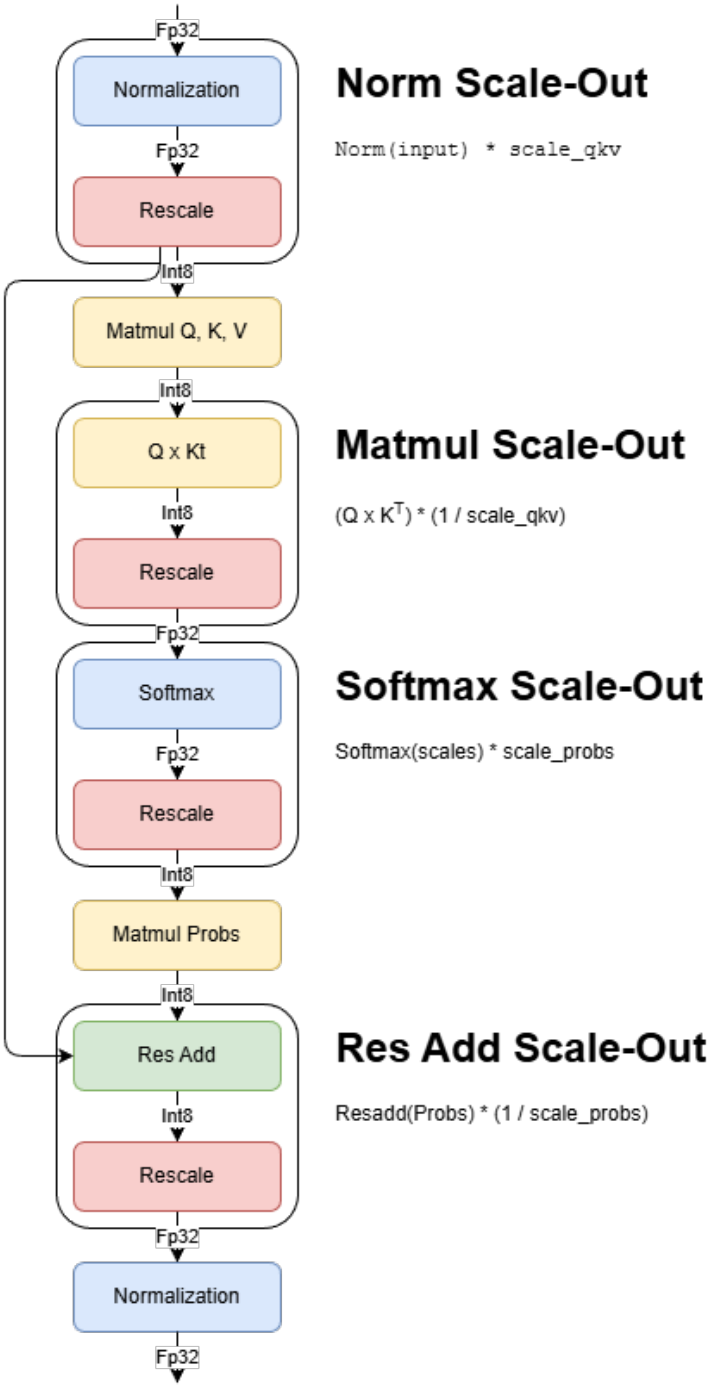


Transformer, incluindo as projeções de autoatenção (Q, K, V) e as redes *Feed-Forward* nos blocos codificadores, são mantidas fixas durante todo o treinamento. Apenas a última camada linear, responsável pela classificação, é substituída para refletir a nova dimensionalidade de classes do *dataset* alvo e submetida à otimização. Essa mecânica assegura que os padrões visuais obtidos através do processo de treinamento em *datasets* de mais larga escala sejam mantidos (LIU et al., 2025).

4.3 Esquema de Quantização Proposta

Com base nas classificações apresentadas na Seção 2.5.1, a estratégia de quantização adotada neste trabalho foi desenhada para equilibrar as restrições de memória do acelerador com a sensibilidade numérica intrínseca da arquitetura *Vision Transformer*. Neste trabalho, empregamos a Quantização Pós-Treinamento (PTQ) operando com precisão mista: as matrizes de pesos de todas as operações de multiplicação matricial geral (GEMM) foram quantizadas para inteiros de 8 bits (INT8), enquanto as ativações foram mantidas no padrão de ponto flutuante de precisão simples (IEEE 754 FP32), assim como descrito na Figura 4.4.

Figura 4.4: Esquema de quantização de precisão mista (W8A32) adotada



A escolha pela manutenção das ativações em alta precisão (estratégia W8A32) é fundamentada no comportamento estatístico das redes baseadas em atenção. Diferentemente das redes convolucionais clássicas, funções como *Softmax* e *GELU*, amplamente presentes nos blocos de Multi-Head Self-Attention (MHSA) e nas redes *Feed-Forward* (FFN) dos ViTs, geram valores de ativação com distribuições altamente irregulares e não-gaussianas, como descrito na Seção 2.5.2. A tentativa de aplicar quantização uniforme de 8 bits sobre esses tensores, que contêm *outliers* significativos, resulta frequentemente em uma degradação da acurácia do modelo. Dessa forma, a abordagem aplicada garante uma redução de aproximadamente 75% no custo de armazenamento dos parâmetros estruturais do modelo, ao mesmo tempo que preserva a integridade funcional e a precisão da inferência.

Em relação à granularidade, implementou-se uma quantização por tensor (*per-tensor*). Nessa abordagem de granularidade grossa, o único parâmetro de quantização, um fator de escala (*scale*), é calculado e aplicado para toda a matriz de pesos de uma respectiva camada. A opção por quantizar ao nível do tensor, em detrimento da quantização por canal (*per-channel*), foi definida para maximizar a compatibilidade e a eficiência no hardware *Gemmini*. Ao utilizar apenas um fator de escala por matriz, simplifica-se o código C gerado para execução de controle de fluxo de dados na arquitetura sistólica do *Gemmini*, ao utilizar instruções CISC pré-construídas, o que permite uma execução mais eficiente do que se fosse implementada manualmente apenas com instruções RISC.

Por fim, a adoção exclusiva da técnica PTQ demonstra-se suficiente para atingir as metas de compressão sem incorrer no elevado custo computacional e de engenharia exigido pelo Treinamento Consciente de Quantização (QAT) utilizando as metodologias de quantização acima descritas.

5 METODOLOGIA EXPERIMENTAL

Este capítulo detalha os procedimentos adotados para a validação funcional e a avaliação de desempenho do hardware proposto. Inicialmente, descreve-se o ambiente de simulação computacional, no qual ferramentas de análise funcional e simuladores acurados ao nível de ciclo são empregados para atestar a correção da inferência no acelerador. Na sequência, aborda-se o fluxo de prototipação física em FPGA, etapa fundamental para a validação empírica da lógica digital do sistema integrado sob condições reais de operação. Por fim, o capítulo detalha a metodologia de implementação direcionada a um fluxo de síntese ASIC para o nó tecnológico de 65nm, o qual fornece as estimativas definitivas de consumo de potência e área de silício utilizadas para a caracterização do coprocessador.

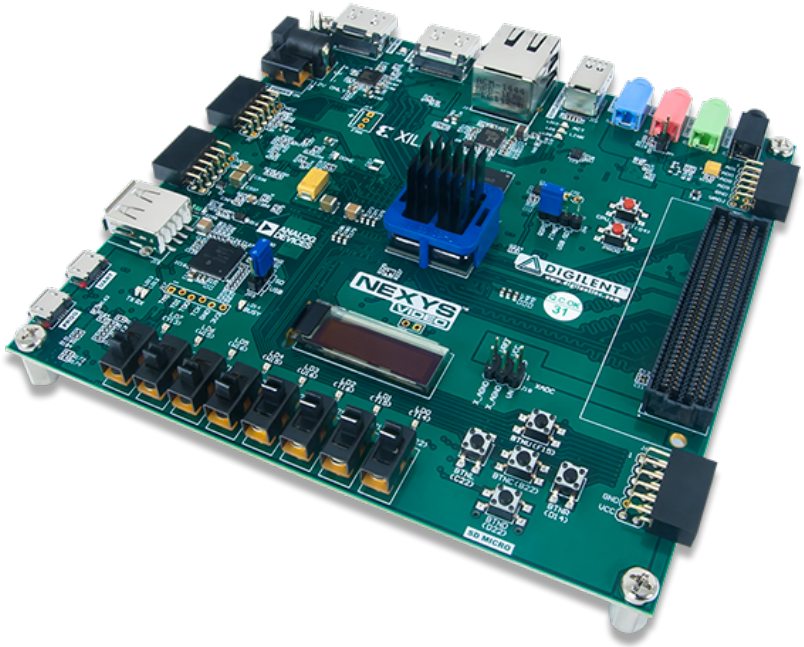
5.1 Simuladores para execução dos experimentos

Para uma avaliação mais rápida das redes, foram utilizados dois simuladores: Spike e Verilator. O primeiro é um simulador funcional da arquitetura RISC-V, portanto reflete o comportamento das instruções, entretanto não simula a execução do hardware, estimando de forma pouco precisa o número de ciclos e tempo de execução. Esse simulador foi utilizado para fornecer exclusivamente informações de acurácia da rede. Por outro lado, o verilator trata-se de um simulador acurado a nível de ciclo, ou seja, reflete o comportamento do hardware desenvolvido em HDL. Nessa etapa também foi utilizada uma camada de tradução: o código HDL (Linguagem de Descrição de Hardware ou Hardware Description Language) para o acelerador foi desenvolvido em Chisel, uma linguagem de descrição de alto nível baseada em Scala. Esse hardware é então convertido em Verilog através de uma camada de tradução.

5.2 Avaliação do desempenho do Acelerador em FPGA

Para assegurar a correção funcional do acelerador em nível lógico e mitigar riscos antes de prosseguir para o complexo fluxo de desenvolvimento físico em silício (ASIC), o sistema foi previamente validado através de um fluxo de prototipação em FPGA (*Field Programmable Gate Array*). Nesta etapa, o *System-on-Chip* (SoC), composto pelo acelerador sistólico acoplado a um núcleo de processamento RISC-V (conforme detalhado

Figura 5.1: Imagem ilustrativa da placa Xilinx Nexys Video



na Seção 2.7), foi submetido ao processo de síntese lógica. Esse procedimento traduziu a descrição RTL (*Register Transfer Level*) do hardware em uma *netlist* estrutural.

Uma vez gerada, a *netlist* foi mapeada e implementada utilizando os blocos construtivos físicos disponíveis no dispositivo programável, o que inclui a alocação de *Look-Up Tables* (LUTs) para a lógica combinacional, *Flip-Flops* para os elementos de memória sequencial, memórias em bloco (BRAMs) para o armazenamento on-chip e módulos de processamento digital de sinais (DSPs) para as operações aritméticas densas. O hardware escolhido para realizar esta validação foi a placa de desenvolvimento Nexys Video, manufaturada pela Xilinx, a qual está ilustrada na Figura 5.1.

Para o *deployment*, foi utilizado o software Xilinx Vivado, o qual disponibiliza ferramentas como o Gerenciador de Hardware, utilizado para reprogramar a FPGA com um novo *bitstream*, ou seja, uma nova configuração de hardware, bem como as ferramentas para disponibilizar a síntese e implementação.

Cabe ressaltar que a prototipação em FPGA serviu primariamente como um veículo para a extração de métricas de desempenho preliminares, como latência e vazão de inferência, além de comprovar a execução correta do modelo quantizado no hardware físico. Dessa forma, as otimizações intrínsecas ao fluxo de implementação de FPGAs transcendem o escopo de análise deste trabalho, cujo foco arquitetural se dá na avaliação do fluxo ASIC.

5.3 Implementação Física (Fluxo ASIC)

A implementação física do sistema foi desenvolvida a partir de um fluxo de projeto de circuitos integrados de aplicação específica (ASIC), direcionado ao nó tecnológico de 65nm da TSMC. O processo de síntese foi conduzido utilizando o conjunto de ferramentas da fabricante Cadence.

Antes da síntese, a ferramenta de geração do acelerador Gemmini foi parametrizada para criar um coprocessador compatível com os modelos propostos. Esse procedimento exigiu o redimensionamento das memórias SRAM internas, especificamente o *Scratchpad* e o *Accumulator*. A Tabela 5.1 descreve a arquitetura desenvolvida para o fluxo ASIC.

Tabela 5.1: Características do Co-processador gemmini sintetizadas

Parâmetro Arquitetural	**Configuração**
Tipos de Dados (Datatypes)	
Entrada e Pesos (Input/Weight)	SInt (8 bits)
Acumulador (Accumulator)	SInt (32 bits)
Saída do Arranjo Espacial	SInt (20 bits)
Arranjo Espacial (Spatial Array)	
Dimensões da Malha (Mesh)	8×8 (Linhas $\times$ Colunas)
Fluxo de Dados (Dataflow)	Weight Stationary
Memória Interna	
Scratchpad (SPAD)	256 KB (4 Bancos, Single-ported)
Acumulador	64 KB (2 Bancos, Multi-ported)
Estruturas de Controle e Pipeline	
Reservation Station (LD / EX / ST)	8 / 16 / 4 entradas
Instruction Queues (LD / EX / ST)	8 / 8 / 2 entradas
Acesso à Memória (DMA e TLB)	
Requisições de Memória em Voo (Máx)	64
Largura do Barramento DMA	128 bits
Tamanho da TLB	4 entradas

A etapa de síntese lógica operou recebendo como entrada o código em linguagem de descrição de hardware (HDL) do coprocessador e a biblioteca de tecnologia alvo. A ferramenta Cadence Genus foi utilizada para converter a descrição em nível de transferência de registro (RTL) em uma *netlist* ao nível de portas lógicas (*gate-level netlist*).

Devido a limitações no acesso direto ao compilador de memórias da TSMC para a tecnologia de 65nm, as instâncias de memória foram tratadas como caixas pretas (*black-boxes*) pela ferramenta de síntese. Após a geração da *netlist*, a contribuição física e ener-

gética das memórias foi calculada isoladamente utilizando métricas documentadas em (WU et al., 2010) para compiladores SRAM de baixo consumo implementados na mesma tecnologia de 65nm. Por fim, esses valores estimados foram diretamente somados aos resultados pós-síntese da lógica digital, compondo os números definitivos referentes à área total do núcleo e à dissipação de potência do acelerador. O fluxo avaliou o comportamento e os requisitos físicos do projeto operando sob uma frequência de relógio de 100 MHz.

6 RESULTADOS EXPERIMENTAIS E DISCUSSÃO

Com base na metodologia descrita na Seção 5, este capítulo apresenta e discute os resultados obtidos sob as diferentes métricas que avaliam o comportamento dos aceleradores baseados em *Vision Transformers* para dispositivos de borda desenvolvidos neste trabalho. A avaliação é estruturada através de parâmetros que quantificam:

- a acurácia das redes nas tarefas de inferência em classificação;
- a dimensão e requisitos de memória das redes quantizadas;
- a aceleração computacional promovida pelo coprocessador Gemmini;
- o desempenho relativo em comparação a dispositivos baseados em GPUs;
- o custo de implementação física, mensurado em área de silício e consumo energético.

Esses parâmetros foram selecionados de forma a avaliar todos os pontos que devem ser considerados para dispositivos de borda ao realizar tarefas de computação visual, formando os seguintes pilares fundamentais para a execução eficiente: (1) acurácia, (2) latência e (3) energia.

6.1 Performance de Classificação e *Footprint* de Memória

Com relação às redes desenvolvidas para o acelerador e descritas na Seção 4.1, foram avaliados os aspectos de precisão nas tarefas de classificação e também o impacto dos novos modelos nos requisitos de memória.

6.1.1 Acurácia das redes

O primeiro aspecto avaliado para a validação do sistema é a acurácia final das redes quantizadas. Este experimento mensura a degradação da precisão decorrente da adoção da quantização pós-treinamento (PTQ). Para tal, compara-se o desempenho do conjunto de teste executado no acelerador Gemmini (cujos pesos operam em precisão INT8) contra uma *baseline* processada em precisão de ponto flutuante padrão (FP32) em uma GPU comercial. Cabe destacar que, para o *baseline*, o hardware específico da GPU é irrelevante, desde que assegure a integridade das operações em FP32.

A Tabela 6.1 consolida os resultados comparativos. Conforme o esperado, a aplicação do esquema PTQ resultou em uma redução da acurácia devido à compressão numérica. No entanto, a estratégia adotada conteve a degradação em patamares aceitáveis para aplicações de borda. Observou-se uma queda máxima de acurácia de 3,43% para tarefas complexas empregando modelos maiores (como o DeiT-Large avaliado na base TinyImageNet), e reduções ainda menores, na ordem de 2,03%, para modelos restritos como o MiniViT-Tiny operando sobre o conjunto CIFAR-10.

6.1.2 Requisitos de memória para execução dos modelos

Em paralelo à acurácia, a viabilidade da inferência em dispositivos de borda é ditada pelo *footprint* de memória (*memory footprint*) dos modelos. Conforme discutido na Seção 2.4, acessos recorrentes à memória de hierarquia superior do SoC (como a Cache L2, na Figura 2.10) ou a memórias externas impõem severas penalidades de latência. Portanto, é imperativo que os tensores processados a cada iteração sejam acomodados integralmente no *Scratchpad* do acelerador.

Tabela 6.1: Comparação entre a acurácia da execução dos modelos quantizados em INT8 (usando PTQ) contra a execução em FP32 na GPU

Dataset	**Model**	**Version**	**Precision**	**Top-1 Acc (%)**	**Accuracy Drop**
TinyImageNet	DeiTViT	Large	FP32	58.13%	-
	DeiTViT	Large	INT8	54.70%	3.43%
CIFAR-10	MiniViT	Tiny	FP32	79.43%	-
	MiniViT	Tiny	INT8	77.40%	2.03%

A Tabela 6.2 também evidencia o impacto estrutural da quantização. A conversão das matrizes de pesos para inteiros de 8 bits propiciou uma compactação média de 3,95x no tamanho total do modelo em comparação à sua versão em ponto flutuante. Essa redução (que encolheu modelos de 83MB para 21MB, por exemplo) mitigou consideravelmente o gargalo de largura de banda, garantindo que o volume de transferência de dados permanecesse compatível com a capacidade de armazenamento interno do hardware dedicado projetado, ao mesmo tempo que reduziu o tempo de acesso à memória entre operações em tensores.

Tabela 6.2: Comparação entre a dimensão dos modelos quantizados em INT8 comparado aos modelos FP32 na GPU

Dataset	Model	Version	Precision	Model Size	Size Reduction
TinyImageNet	DeiTViT	Large	FP32	83MB	-
	DeiTViT	Large	INT8	21MB	3.95×
CIFAR-10	MiniViT	Tiny	FP32	1.9MB	-
	MiniViT	Tiny	INT8	513kB	3.70×

6.1.3 Avaliação das diferentes redes desenvolvidas

Outro aspecto avaliado é o impacto arquitetural da introdução do *token* de destilação (presente na arquitetura DeiT) para o treinamento em conjuntos de dados reduzidos. A Figura 6.1 apresenta a análise comparativa da acurácia *Top-1* entre as redes MiniViT (padrão) e DeiT operando em quatro escalas paramétricas distintas (*Tiny*, *Small*, *Base* e *Large*). O gráfico contrasta a execução no acelerador Gemmini, sujeito à quantização INT8, com a linha de base processada em GPU sob precisão FP32.

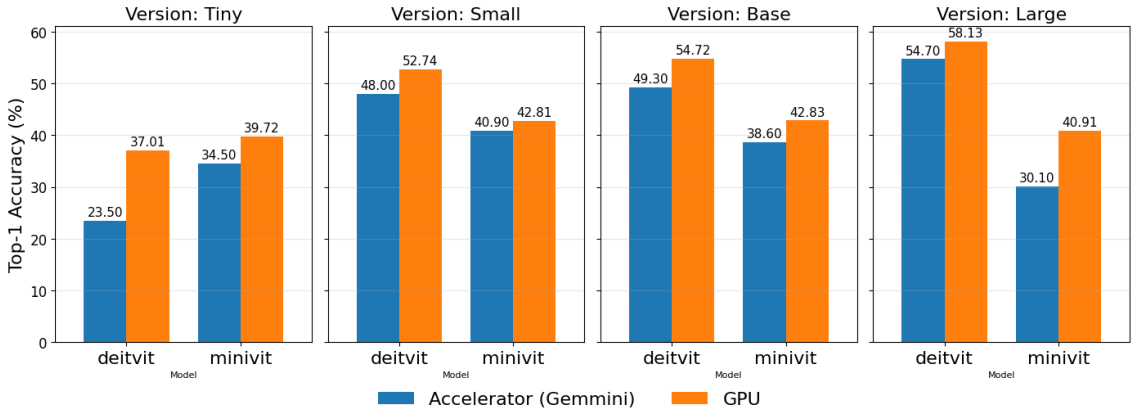
Os resultados demonstram que o *token* de destilação confere uma vantagem representativa que escala proporcionalmente à capacidade da rede. Nas configurações *Small*, *Base* e *Large*, os modelos da família DeiT superaram as variantes MiniViT em ambas as plataformas de hardware. Destaca-se a versão *Large* executada no acelerador sistólico, onde o modelo DeiT atingiu 54,70% de acurácia, em contraste aos 30,10% obtidos pelo modelo MiniViT equivalente.

Por fim, o experimento evidencia que todas as arquiteturas avaliadas sofrem uma redução de acurácia ao transitar da representação em ponto flutuante (GPU) para o processamento em inteiros (Gemmini). Esse comportamento reforça o desafio arquitetural de preservar a integridade matemática dos mecanismos de atenção sob os efeitos de técnicas de quantização em hardware.

6.2 Aceleração do Hardware

Outro ponto a ser considerado é o quanto a execução de redes neurais é beneficiada pela operação com o co-processador Gemmini. Nesse contexto, para realizar essa avaliação o SoC Gemmini foi submetido à uma série de testes utilizando apenas o processador RISC-V Rocket, com suporte à instruções de multiplicação e utilizando os algoritmos aproximativos descritos na Seção 4.1.3. Posteriormente, o mesmo teste foi executado

Figura 6.1: Comparação entre os modelos MiniDeiT e MiniViT



utilizando o acelerador Gemmini.

Após essas execuções, o número de ciclos foi computado e multiplicado pela frequência de operação obtida na síntese ASIC, que será apresentada na seção 6.3.1. Os resultados obtidos estão dispostos na Tabela 6.3.

Tabela 6.3: Comparação de Performance: Pesos Estacionários vs. CPU (MiniViT com CIFAR-10)

Versão ViT	**Alvo**	**Ciclos Totais**	**Matrizes**	**Normalizações**	**Aceleração**
Tiny	WS	6.83 M	89.63 K	6.74 M	∼36.2x
	CPU	247.58 M	238.49 M	9.08 M	
Small	WS	15.30 M	133.89 K	15.17 M	∼51.1x
	CPU	782.25 M	761.80 M	20.44 M	
Base	WS	23.71 M	172.59 K	23.54 M	∼75.3x
	CPU	1.78 B	1.75 B	32.92 M	
Large	WS	63.16 M	369.91 K	62.78 M	∼161.1x
	CPU	10.17 B	10.08 B	91.22 M	

A Tabela 6.3 demonstra o impacto da adoção de aceleradores dedicados na execução de modelos de aprendizado profundo. Ao transferir a carga de trabalho computacional da CPU para um acelerador dedicado, observa-se um ganho de desempenho que escala de forma diretamente proporcional à dimensionalidade da rede. Enquanto o modelo mais restrito (MiniViT-Tiny) apresenta uma aceleração de 36,2 vezes, a versão mais robusta (MiniViT-Large) atinge um ganho substancial de 161,1 vezes. Nesse cenário, o tempo total de inferência sofre uma redução de 10,17 bilhões de ciclos na CPU para apenas 63,16 milhões de ciclos no acelerador.

A raiz dessa aceleração expressiva reside na mitigação do gargalo imposto pelas

multiplicações matriciais. A execução dessas operações possui complexidade $\mathcal{O}(n^3)$, de modo que sua execução em CPU consome muito tempo de processamento. O acelerador anula essa restrição através do processamento altamente paralelo e da maximização do reúso espacial e temporal de dados, características intrínsecas à topologia sistólica, conforme Seção 2.6. Essa característica demonstra que o acelerador é capaz de utilizar uma matriz bidimensional de Elementos de Processamento (PEs) para tornar a operação de multiplicação de matrizes em um processamento de complexidade próxima a $\mathcal{O}(n)$.

Consequentemente, ao analisarmos a penúltima coluna, percebemos que as normalizações e ativações não lineares (*Softmax*, *GELU* e *Layernorm*) se tornam o gargalo do sistema, uma vez que estas ainda estão sendo processadas em CPU nesta análise. Nota-se que essas operações passaram a consumir mais de 98% dos ciclos totais.

6.3 Eficiência do Hardware

O próximo pilar a ser analisado, a fim de assegurar uma execução eficiente na borda, é a eficiência energética e o custo em área de silício, uma vez que essas características impactam altamente na manufaturabilidade e na possibilidade de execução na borda com uso de baterias.

6.3.1 Resultados do fluxo ASIC para TSMC 65nm

A avaliação da arquitetura proposta, sintetizada sob o nó tecnológico de 65nm da TSMC, resultou em uma área total do acelerador Gemmini estruturalmente contida em 2,77 mm^2. O consumo de potência estimado para o arranjo sistólico foi de 149,79 mW operando a uma frequência de 100 MHz. Os resultados obtidos estão sintetizados na Tabela 6.4.

Tabela 6.4: Hardware Resource and Efficiency Comparison

Metric	**Tegra X1 GPU**	**Proposed ASIC**	**Estimated Scaled ASIC**
Process Node	20nm	65nm	20nm
Frequency	1000 MHz	100 MHz	100 MHz
Power (W)	10 W	149.79 mW	17.92 mW
Area (mm^2)	-	2.77	0.388

6.3.2 Comparação com dispositivos comerciais

Para estabelecer uma base de comparação com plataformas *off-the-shelf* comumente adotadas na borda, o acelerador foi contrastado contra a GPU NVIDIA Tegra X1, presente na plataforma Jetson Nano. Tendo em vista que o processador gráfico comercial opera em um nó de 20nm com um consumo energético de aproximadamente 10 W, tornou-se necessária a projeção do circuito desenvolvido para garantir uma equivalência tecnológica. Essa projeção foi realizada a partir do estudo presente em (STILLMAKER; BAAS, 2017).

Ao escalar as métricas de pós-síntese do Gemmini para litografia de 20nm, o *design* atinge uma predição de área de 0,388 mm^2 e um consumo mitigado de apenas 17,92 mW. Essa arquitetura dedicada refletiu em ganhos substanciais de eficiência energética, avaliada primariamente através da métrica de Joules consumidos por imagem processada (J/imagem). A eliminação da sobrecarga inerente à decodificação de instruções de propósito geral das GPUs proporcionou ao fluxo de *hardware-software co-design* proposto uma melhoria de 3,95x nas inferências por $mm^2 \cdot J$. Demonstra-se, assim, a superioridade da adoção de matrizes sistólicas conjugadas a esquemas restritos de quantização para inferências de *Vision Transformers* embarcados.

7 CONCLUSÕES

A execução de tarefas de computação visual em dispositivos de borda constitui uma área de pesquisa ativa na literatura recente, impulsionada pela necessidade de aplicar modelos de alta acurácia sob restrições físicas severas. Embora os Vision Transformers (ViTs) apresentem desempenho proeminente, sua implementação em cenários embarcados enfrenta desafios: a alta complexidade computacional e a dificuldade de treinamento do mecanismo de atenção em conjuntos de dados reduzidos, devido à ausência de vieses indutivos. Este trabalho demonstrou que a integração de técnicas de Destilação de Conhecimento, especificamente através do uso de um *token* de destilação dedicado, resolve a limitação do aprendizado em bases de dados reduzidas, permitindo a convergência e viabilizando o uso de estruturas miniaturizadas.

Quanto ao hardware, as plataformas de processamento convencionais impõem barreiras para a execução dessas redes na borda. Enquanto a execução exclusiva em CPUs demonstra-se lenta para lidar com operações sobre tensores, o emprego de unidades de processamento gráfico (GPUs) comerciais fornece alta vazão de dados ao custo de um consumo energético proibitivo para sistemas alimentados por baterias. Nesse contexto, os aceleradores baseados em arranjos sistólicos consolidam-se como uma alternativa de alta eficiência. Ao mapear de forma paralela e espacial as operações de multiplicação matricial (GEMM), essa arquitetura reduz o principal gargalo do modelo, transformando a computação de multiplicações de matrizes de ordem de complexidade $\mathcal{O}(N^3)$ em uma execução $\mathcal{O}(N)$.

Os resultados obtidos demonstram que a eficiência energética e computacional em sistemas integrados depende do co-design entre hardware e software. A adoção de uma estratégia de quantização pós-treinamento (PTQ) de precisão mista permitiu uma redução significativa do tamanho e tempo de execução do modelo. Uma vez que a representação numérica das distribuições não-gaussianas e os outliers das funções de ativação e normalização dos ViTs sofrem severa degradação sob esquemas de quantização em inteiro de 8 bits, a manutenção das ativações em FP32 preservou a integridade do modelo. Para o ambiente de borda, a redução do custo computacional compensou a perda marginal de precisão, viabilizando uma compactação de até 3,95x no tamanho do modelo com uma degradação de acurácia limitada a menos de 3,43%.

Por fim, a aceleração das multiplicações matriciais no arranjo sistólico mitigou drasticamente o tempo de execução dos modelos. Com a redução drástica nos ciclos de

processamento das GEMMs, as operações de suporte e normalização (como LayerNorm, Softmax e GELU) passaram a dominar mais de 98% do tempo de execução do coprocessador. Este fenômeno indica que os desenvolvimentos futuros de hardware-software co-design para arquiteturas baseadas em atenção devem concentrar-se na criação de unidades aritméticas dedicadas à aceleração nessas etapas específicas de normalização.

REFERÊNCIAS

AMID, A. et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs. **IEEE Micro**, v. 40, n. 4, p. 10–21, 2020. ISSN 1937-4143.

BA, J. L.; KIROS, J. R.; HINTON, G. E. **Layer Normalization**. 2016. Available from Internet: <https://arxiv.org/abs/1607.06450>.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: **IEEE Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2009. p. 248–255. Available from Internet: <https://www.image-net.org/index.php>.

DOSOVITSKIY, A. et al. **An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale**. 2021. Available from Internet: <https://arxiv.org/abs/2010.11929>.

GENC, H. et al. **Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration**. 2021. Available from Internet: <https://arxiv.org/abs/1911.09925>.

GHOLAMI, A. et al. **A Survey of Quantization Methods for Efficient Neural Network Inference**. 2021. Available from Internet: <https://arxiv.org/abs/2103.13630>.

GOOGLE. **Google Coral NPU (GitHub)**. 2026. Available from Internet: <https://github.com/google-coral/coralnpu>.

HAO, Z. et al. **Learning Efficient Vision Transformers via Fine-Grained Manifold Distillation**. 2022. Available from Internet: <https://arxiv.org/abs/2107.01378>.

HE, K. et al. **Deep Residual Learning for Image Recognition**. 2015. Available from Internet: <https://arxiv.org/abs/1512.03385>.

HENDRYCKS, D.; GIMPEL, K. **Gaussian Error Linear Units (GELUs)**. 2023. Available from Internet: <https://arxiv.org/abs/1606.08415>.

HOWARD, A. G. et al. **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**. 2017. Available from Internet: <https://arxiv.org/abs/1704.04861>.

HUANG, M. et al. An integer-only and group-vector systolic accelerator for efficiently mapping vision transformer on edge. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 70, n. 12, p. 5289–5301, 2023.

JORDAN, J. **Neural networks: representation**. 2017. Acesso em: 5 jun. 2026. Available from Internet: <https://www.jeremyjordan.me/intro-to-neural-networks/>.

JOUPPI, N. P. et al. **In-Datacenter Performance Analysis of a Tensor Processing Unit**. 2017. Available from Internet: <https://arxiv.org/abs/1704.04760>.

KIM, S. et al. **I-BERT: Integer-only BERT Quantization**. 2021. Available from Internet: <https://arxiv.org/abs/2101.01321>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2012. v. 25. Available from Internet: <https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

KUNG. Why systolic architectures? **Computer**, v. 15, n. 1, p. 37–46, 1982.

LI, Z.; GU, Q. **I-ViT: Integer-only Quantization for Efficient Vision Transformer Inference**. 2023. Available from Internet: <https://arxiv.org/abs/2207.01405>.

LI, Z. et al. **Auto-ViT-Acc: An FPGA-Aware Automatic Acceleration Framework for Vision Transformer with Mixed-Scheme Quantization**. 2022. Available from Internet: <https://arxiv.org/abs/2208.05163>.

LIU, T. et al. **Sparse-Tuning: Adapting Vision Transformers with Efficient Fine-tuning and Inference**. 2025. Available from Internet: <https://arxiv.org/abs/2405.14700>.

LIU, Z. et al. **Swin Transformer: Hierarchical Vision Transformer using Shifted Windows**. 2021. Available from Internet: <https://arxiv.org/abs/2103.14030>.

LUO, Y. et al. **An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning**. 2025. Available from Internet: <https://arxiv.org/abs/2308.08747>.

MEHTA, S.; RASTEGARI, M. **MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer**. 2022. Available from Internet: <https://arxiv.org/abs/2110.02178>.

NAG, S. et al. Vita: A vision transformer inference accelerator for edge applications. In: **2023 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2023. p. 1–5.

NAGEL, M. et al. **A White Paper on Neural Network Quantization**. 2021. Available from Internet: <https://arxiv.org/abs/2106.08295>.

NVIDIA. **NVIDIA Faster Transformer (GitHub)**. 2023. Available from Internet: <https://github.com/NVIDIA/FasterTransformer>.

PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization and Design: The Hardware/Software Interface**. Risc-v. [S.l.]: Morgan Kaufmann, 2020. ISBN 978-0128203316.

RIDNIK, T. et al. **ImageNet-21K Pretraining for the Masses**. 2021. Available from Internet: <https://arxiv.org/abs/2104.10972>.

SCHRAUDOLPH, N. N. A fast, compact approximation of the exponential function. **Neural computation**, MIT Press, v. 11, n. 4, p. 853–862, 1999.

STILLMAKER, A.; BAAS, B. Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm. **Integration**, v. 58, p. 74–81, 2017. ISSN 0167-9260. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0167926017300755>.

TAN, M.; LE, Q. V. **EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks**. 2020. Available from Internet: <https://arxiv.org/abs/1905.11946>.

TOUVRON, H. et al. **Training data-efficient image transformers distillation through attention**. 2021. Available from Internet: <https://arxiv.org/abs/2012.12877>.

VASWANI, A. et al. **Attention Is All You Need**. 2023. Available from Internet: <https://arxiv.org/abs/1706.03762>.

WANG, T. et al. Via: A novel vision-transformer accelerator based on fpga. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 41, n. 11, p. 4088–4099, 2022.

WU, K. et al. **TinyViT: Fast Pretraining Distillation for Small Vision Transformers**. 2022. Available from Internet: <https://arxiv.org/abs/2207.10666>.

WU, S. et al. A 65nm embedded low power sram compiler. In: **13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems**. [S.l.: s.n.], 2010. p. 123–124.

XIAO, G. et al. **SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models**. 2024. Available from Internet: <https://arxiv.org/abs/2211.10438>.

YOU, H. et al. **ViTCoD: Vision Transformer Acceleration via Dedicated Algorithm and Accelerator Co-Design**. 2025. Available from Internet: <https://arxiv.org/abs/2210.09573>.

YUAN, Z. et al. Ptq4vit: Post-training quantization framework for vision transformers. **CoRR**, abs/2111.12293, 2021. Available from Internet: <https://arxiv.org/abs/2111.12293>.

ZHAI, X. et al. **Scaling Vision Transformers**. 2022. Available from Internet: <https://arxiv.org/abs/2106.04560>.