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

ÁTILA UEBEL

Explorando a Descentralização para Melhorar o Desempenho das Redes Neurais no Contínuo Heterogêneo entre Dispositivo-Móvel-Borda-Nuvem

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

Orientador: Prof. Dr. Antônio Carlos S. Beck Filho

Porto Alegre
2025

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

1 AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão a todos que estiveram ao meu lado durante esta jornada.

Aos meus pais, que sempre me ofereceram suporte e apoio incondicional, minha eterna admiração e agradecimento. Sem o carinho e os ensinamentos deles, nada disso seria possível.

À minha família, incluindo avós, dindos, tios e todos os parentes, que sempre estiveram prontos para ajudar de todas as formas que podiam, minha imensa gratidão. O apoio de vocês foi fundamental em cada passo dessa trajetória.

Aos meus amigos antigos e aos novos que fiz ao longo dos anos na faculdade, especialmente ao grupo SETE, que tornaram tanto os momentos acadêmicos quanto a minha vida mais alegre, leve e cheia de boas lembranças. A amizade de vocês fez toda a diferença!

A todos os amigos e conhecidos que me ofereceram caronas e me permitiram ver meus pais com maior frequência, meu sincero agradecimento. Esses gestos fizeram toda a diferença na minha vida pessoal e acadêmica.

Gostaria também de agradecer à UL Solutions, que gentilmente cedeu o acesso às suas ferramentas e facilitou significativamente o meu trabalho. O suporte fornecido foi de grande importância para o desenvolvimento deste projeto.

Por fim, à minha faculdade, e especialmente ao meu orientador, Antônio Carlos Beck Filho, por todo o aprendizado, orientação e apoio. A oportunidade de trabalhar com você foi um privilégio, e sua dedicação foi essencial para que eu pudesse concluir esta fase tão importante da minha vida.

A todos que de alguma forma contribuíram para essa caminhada, meu muito obrigado. Cada um de vocês teve um papel fundamental na realização deste momento tão especial.

RESUMO

As redes neurais e o aprendizado profundo se tornaram essenciais para uma ampla gama de aplicações de Inteligência Artificial (IA), abrangendo áreas como reconhecimento de imagens, processamento de linguagem natural e sistemas autônomos parciais ou totais. No entanto, a implementação desses modelos em dispositivos com capacidade computacional limitada, como dispositivos móveis, representa um desafio significativo.

Este trabalho se propõe, inicialmente, a introduzir as principais camadas de execução relacionadas ao processamento de redes neurais em dispositivos móveis: **processamento local no dispositivo móvel**, **processamento na borda** e **processamento na nuvem**. Essas camadas representam diferentes níveis de infraestrutura disponíveis para executar as tarefas computacionais. O **processamento local** ocorre diretamente no dispositivo móvel, oferecendo maior privacidade e menor dependência de conectividade, mas é limitado pela capacidade de hardware. O **processamento na borda**, realizado em servidores próximos ao dispositivo, busca reduzir a latência e melhorar a eficiência, aproveitando recursos intermediários entre o dispositivo e a nuvem. Já o **processamento na nuvem** disponibiliza poder computacional elevado e escalabilidade, mas sofre geralmente com maior latência devido à distância física e à dependência de rede. A existência dessas camadas é motivada pela necessidade de balancear três fatores fundamentais: **latência**, **capacidade de processamento** e **eficiência energética**. Dispositivos móveis são limitados em capacidade computacional e consumo de energia, mas demandas modernas, como redes neurais profundas, exigem grandes volumes de processamento. Dessa forma, as camadas permitem escolher a melhor infraestrutura para executar cada tarefa, dependendo dos requisitos específicos de cada aplicação.

Nesse contexto, será explorado o conceito de *offloading*, que consiste na transferência estratégica de cargas computacionais entre essas camadas para otimizar a execução das redes neurais. Existem diversas abordagens de *offloading*, que envolvem diferentes localizações de execução, incluindo a execução no próprio dispositivo, o *offloading* para servidores de borda, o *offloading* para servidores na nuvem, ou ainda a partição da carga de trabalho entre essas opções. No entanto, o foco principal deste estudo será realizar uma série de testes em um cenário simulado envolvendo o processamento total nos dispositivos móveis, servidores de borda e nuvem. Esses testes, juntamente com os dados de latência de rede, visam identificar "o melhor local de execução" para cada tipo de cenário, otimizando assim o desempenho das redes neurais.

Ao longo deste trabalho, será demonstrado como a escolha de uma estratégia adequada de *offloading* é essencial para superar as limitações computacionais de dispositivos móveis, otimizando o desempenho das redes neurais e oferecendo uma solução eficiente para cenários com restrições de hardware.

Palavras-chave: Redes neurais. aprendizado profundo. offloading. dispositivos móveis. dispositivos de borda. servidores na nuvem. partição de carga de trabalho. tempo de execução. consumo de energia. otimização de desempenho.

ABSTRACT

Deep neural networks and deep learning have become essential for a wide range of Artificial Intelligence (AI) applications, including image recognition, natural language processing, and partially or fully autonomous systems. However, implementing these models on devices with limited computational capacity, such as mobile devices, represents a significant challenge.

This work initially aims to introduce the main execution layers related to neural network processing on mobile devices: **local processing on the mobile device**, **edge processing**, and **cloud processing**. These layers represent different levels of infrastructure available to perform computational tasks. **Local processing** occurs directly on the mobile device, offering greater privacy and less dependence on connectivity but limited by hardware capacity. **Edge processing**, carried out on servers close to the device, aims to reduce latency and improve efficiency by leveraging intermediate resources between the device and the cloud. Meanwhile, **cloud processing** provides high computational power and scalability but often suffers from higher latency due to physical distance and network dependency. These layers exist to balance three fundamental factors: **latency**, **processing capacity**, and **energy efficiency**. Mobile devices are constrained in computational capacity and energy consumption, yet modern demands such as deep neural networks require significant processing volumes. Thus, these layers allow selecting the best infrastructure to execute each task depending on the specific requirements of each application.

In this context, the concept of offloading will be explored, which consists of strategically transferring computational loads between these layers to optimize neural network execution. Various offloading approaches involve different execution locations, including execution on the device itself, offloading to edge servers, offloading to cloud servers, or even partitioning the workload among these options. However, the primary focus of this study will be conducting a series of tests in a simulated scenario involving complete processing on mobile devices, edge servers, and cloud servers. These tests, along with network latency data, aim to identify “the best execution location” for each type of scenario, thereby optimizing the performance of neural networks.

Throughout this work, it will be demonstrated how choosing an appropriate offloading strategy is essential to overcome the computational limitations of mobile devices, optimiz-

ing the performance of neural networks and providing an efficient solution for hardware-constrained scenarios.

Keywords: Neural networks, deep learning, offloading, mobile devices, edge devices, cloud servers, workload partitioning, execution time, power consumption, performance optimization..

LISTA DE ABREVIATURAS E SIGLAS

AE Autoencoder

CNN Convolutional Neural Network

DNN Deep Neural Network

DP Deep Learning

FCNN Fully Connected Neural Network

GAN Generative Adversarial Network

IA Inteligência Artificial

LSTM Long Short-Term Memory

PLN Processamento de Linguagem Natural

RNN Recurrent Neural Network

RTT Round-Trip Time

SDK Software Development Kit

VC Visão Computacional

KD Knowledge Distillation

GPU Unidades de Processamento Gráfico

FPGA Field Programmmable Gate Array

ASIC Circuito Integrado de Aplicação Específica

NNAPI Android Neural Networks API

TPU Unidades de Processamento Tensorial

NPU Unidades de Processamento Neural

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

2 INTRODUÇÃO

A rápida evolução da tecnologia tem impulsionado avanços significativos no campo do aprendizado de máquina, particularmente no contexto do aprendizado profundo. Nos últimos anos, o aprendizado profundo tem demonstrado um sucesso notável em uma ampla gama de domínios de aplicação, incluindo visão computacional, processamento de linguagem natural e análise de big data. Um marco crucial nessa trajetória foi a consistente superação de métodos tradicionais pelo aprendizado profundo no Desafio de Reconhecimento Visual em Escala Grande do ImageNet (ILSVRC) desde 2012 (RUSSAKOVSKY O., 2015). No entanto, este sucesso não vem sem desafios. A alta precisão obtida com o aprendizado profundo está vinculada a exigências significativas de computação e memória. Durante o treinamento, milhões de parâmetros precisam ser refinados iterativamente ao longo de vários períodos de tempo. Já na inferência, a alta dimensionalidade dos dados de entrada (como uma imagem de alta resolução) e os milhões de cálculos necessários sobre esses dados contribuem para essas demandas elevadas.

A abordagem comumente utilizada para lidar com tais requisitos computacionais é aproveitar os recursos oferecidos pela computação em nuvem. Essa solução também enfrenta desafios significativos já que é necessária a transferência de dados da fonte na borda da rede (por exemplo, smartphones e sensores da Internet das Coisas (IoT)) para uma localização centralizada na nuvem, ocasionando problemas como:

1. **Latência:** O atraso no acesso aos serviços em nuvem geralmente não é garantido. Enviar dados para a nuvem para inferência ou treinamento pode incorrer em atrasos adicionais na fila e propagação na rede, não atendendo aos rigorosos requisitos de baixa latência ponta a ponta necessários para aplicativos interativos em tempo real, como direção autônoma cooperativa (KHELIFI et al., 2019). Experimentos do mundo real mostraram que a terceirização de uma transmissão de câmera para um servidor Amazon Web Services e a realização de uma tarefa de visão computacional levam mais de 200 ms de ponta a ponta (SATYANARAYANAN, 2017).
2. **Escalabilidade:** Enviar todos os dados das fontes para a nuvem introduz problemas de gargalo à medida que o número de dispositivos conectados aumenta. Fontes de dados intensivas em largura de banda, como fluxos de vídeo, são particularmente preocupantes.
3. **Privacidade:** Os dados utilizados para aprendizado profundo frequentemente contêm informações pessoais sensíveis, o que pode gerar preocupações entre os usuá-

rios cujos dados e comportamentos serão capturados. Esse cenário resulta em hesitação por parte dos usuários em transferir suas informações para a nuvem, incluindo dados como voz, imagens faciais e digitais. Um exemplo relevante é a implantação de câmeras e outros sensores em espaços públicos na cidade de Nova York, que levantou sérias questões de privacidade entre grupos de defesa dos direitos civis (JEANS, 2019).

4. **Confiabilidade:** Muitos aplicativos baseados em computação em nuvem dependem de comunicações sem fio e redes convencionais para conectar usuários aos serviços. No entanto, em cenários industriais ou críticos, é fundamental que os serviços inteligentes mantenham alta confiabilidade, mesmo diante de falhas ou interrupções na conexão de rede.

Como alternativa para enfrentar tais dificuldades, a computação na borda surge como uma alternativa atraente. Uma fina "malha"de recursos de computação fornece capacidades computacionais mais próximas aos dispositivos finais (SATYANARAYANAN et al., 2009) e traz consigo muitas vantagens, conforme mostrado na Figura 2.1. Certamente, elas não são mutuamente exclusivas (MUDASSAR; KO; MUKHOPADHYAY, 2018; YOUSEFPOUR et al., 2019), mas sim perfeitamente complementares, de modo que a computação na borda estende as capacidades da nuvem.

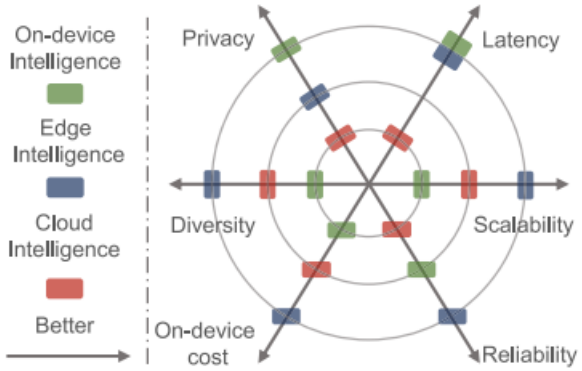
Comparada apenas com a computação em nuvem, a computação na borda traz uma resposta ágil de serviço. Serviços hospedados na borda podem reduzir significativamente o atraso das transmissões de dados e melhorar a velocidade de resposta, ou seja, reduz a latência ponta a ponta e viabiliza serviços em tempo real devido à proximidade aos usuários e utilização da nuvem somente quando necessário (KANG et al., 2017).

Também há alívio da carga de tráfego da rede backbone ao permitir que nós de computação distribuída na borda lidem com muitas tarefas de computação sem a necessidade de trocar dados com a nuvem, possibilitando uma arquitetura hierárquica que inclui dispositivos finais, nós de borda e centros de dados em nuvem, escalando recursos de computação e evitando gargalos de rede em um local central.

Os desafios da privacidade são reduzidos, já que é permitido que os dados sejam analisados perto da fonte, talvez por um servidor de borda local confiável, evitando assim a travessia da Internet pública e reduzindo a exposição a ataques de privacidade e segurança.

Embora a computação de borda ofereça benefícios como menor latência, melhor escalabilidade, maior privacidade e mais confiabilidade, ainda existem desafios significa-

Figura 2.1: Comparação de capacidades de inteligência na nuvem, no dispositivo e na borda



Fonte: (WANG et al., 2020)

tivos para implementar aprendizado profundo nesse contexto. Um dos principais desafios é atender às elevadas demandas de recursos do aprendizado profundo em dispositivos de computação de borda, que geralmente têm capacidades mais limitadas. O aprendizado profundo precisa ser adaptado para ser executado em diversos dispositivos de borda, que variam desde servidores de borda com boa capacidade e equipados com GPUs, até smartphones com processadores móveis e dispositivos Raspberry Pi com recursos bem limitados. O segundo desafio é determinar como os dispositivos de borda devem se coordenar entre si e com a nuvem, considerando as capacidades de processamento variadas e as condições de rede dinâmicas, para garantir um desempenho eficiente de ponta a ponta nas aplicações. Além disso, a privacidade continua sendo um problema, já que alguns dados ainda precisam ser trocados entre os dispositivos de borda e possivelmente com a nuvem.

Diversos pesquisadores têm explorado uma variedade de abordagens, que vão desde o desenvolvimento de hardware até a criação de sistemas para modelagem e análise teórica. O uso de aprendizado profundo para aprimorar a manutenção e o gerenciamento dinâmico e adaptativo na borda da rede também se mostra promissor. Com o avanço da tecnologia de comunicação, os métodos de acesso à rede estão se tornando mais variados. Ao mesmo tempo, a infraestrutura de computação na borda atua como um elo intermediário, proporcionando uma conexão mais confiável e persistente entre os dispositivos finais e a nuvem (YANG, 2019). Dessa forma, os dispositivos finais, a borda e a nuvem estão convergindo gradualmente em uma comunidade de recursos compartilhados.

Considerando os desafios discutidos, visamos contextualizar todo o cenário de computação na borda, e testar algumas formas de divisão de carga na busca do que seria o local ideal de execução conforme o cenário. Analisaremos o desempenho em relação

à execução da rede neural, comparando a execução exclusiva no dispositivo, terceirização completa para a borda e terceirização completa para a nuvem. Em todos os testes, o objetivo principal será alcançar o tempo de execução mais curto possível, considerando também a latência envolvida para a terceirização, visto que ela é uma grande influenciadora no tempo total de execução.

No **Capítulo 2**, somos apresentados aos principais conceitos e termos tanto sobre aprendizado profundo quanto computação em borda, prezando o melhor entendimento do leitor para os capítulos seguintes. O **Capítulo 3**, "Inferência", abordará os aspectos técnicos e práticos da inferência de modelos de aprendizado profundo, destacando técnicas, otimizações e ferramentas específicas para a execução eficiente desses modelos em dispositivos com recursos limitados. No **Capítulo 4**, "Treinamento", será discutido o processo de treinamento de modelos de aprendizado profundo, incluindo estratégias de treinamento distribuído e em borda, bem como os desafios inerentes a esse processo. O **Capítulo 5** será dedicado à "Pesquisa", onde serão apresentadas as metodologias de pesquisa adotadas, os experimentos realizados e o software e hardware utilizados. Os resultados obtidos durante a pesquisa serão detalhados no **Capítulo 6**, "Resultados", oferecendo uma análise compreensiva dos dados coletados, bem como a avaliação da performance dos modelos em cada dispositivo. Finalmente, o **Capítulo 7**, "Conclusão e Considerações Finais", reunirá as principais conclusões do trabalho, discutirá as contribuições para a área de aprendizado profundo na borda e apontará possíveis direções para futuras pesquisas.

3 FUNDAMENTOS E TRABALHO RELACIONADOS

Este artigo abordará técnicas e termos específicos de aprendizado profundo e computação em borda/nuvem, portanto forneceremos primeiramente breve contextualizações tanto sobre as medidas utilizadas até a conceitos mais complexos.

3.1 Aprendizado Profundo

3.1.1 Medição de Desempenho

O aprendizado profundo pode ser aplicado em aprendizado supervisionado e não supervisionado, com métricas de sucesso variando de acordo com o domínio de aplicação. Por exemplo, na detecção de objetos, a precisão é medida pela mAP(RUSSAKOVSKY O., 2015), enquanto na tradução automática, utiliza-se o BLEU(PAPINENI et al., 2002). Métricas gerais como taxa de transferência, latência, energia também são consideradas e estão resumidas na Tabela 3.1.

Tabela 3.1: Métricas de Desempenho Comuns

Métricas de Desempenho
Latência (s)
Energia (mW, J)
Solicitações Simultâneas Atendidas (#)
Largura de Banda (Mbps)
Precisão (específica da aplicação)

Fonte: Os Autores

Escolher ou desenvolver modelos DNN apropriados para uma determinada aplicação é desafiador devido à vasta gama de decisões de hiper parâmetros envolvidas. Entender as compensações entre velocidade, precisão, uso de memória, consumo de energia e outros recursos é essencial para os criadores de modelos e desenvolvedores de aplicativos. Essas informações são frequentemente encontradas em estudos de pesquisa ou em análises independentes(HUANG et al., 2017).

Quando se trata de direcionar dispositivos móveis, é essencial considerar as avaliações de desempenho específicas de CPUs e GPUs móveis. Estudos têm investigado as trocas entre precisão e latência em dispositivos móveis (RAN et al., 2018), como a diminuição da dimensionalidade do tamanho de entrada para aprimorar o desempenho. Modelos DNN especialmente adaptados para dispositivos móveis, como o MobileNets(HOWARD

et al., 2017), apresentam dados de desempenho em termos de operações de multiplicação e adição, facilitando a estimativa de características de latência em diferentes hardwares móveis.

A abordagem do aprendizado de máquina automatizado mostra-se promissora para a seleção de modelos DNN e a otimização de hiper parâmetros. Técnicas como aprendizado por reforço e métodos tradicionais de aprendizado de máquina têm sido sugeridos para a escolha dos hiper parâmetros mais adequados (TAN et al., 2019), especialmente em ambientes de borda, como dispositivos móveis.

3.1.2 Modelos de Aprendizado Profundo

No contexto da Visão Computacional (VC), do Processamento de Linguagem Natural (PLN) e da Inteligência Artificial (IA), o Deep Learning (DL) é amplamente utilizado em diversas aplicações e demonstra um desempenho superior, como mostrado em (LECUN; BENGIO; HINTON, 2015). O aprendizado profundo nada mais é do que um algoritmo de predição, conhecido como modelo, composto por várias camadas. Durante a inferência em aprendizado profundo, os dados de entrada percorrem as camadas em sequência, com cada camada realizando multiplicações de matrizes nos dados. A saída de uma camada serve como entrada para a camada seguinte, e assim por diante. Quando os dados passam pela última camada, a saída final pode ser um recurso ou uma classificação. Quando o modelo possui várias camadas sequenciais, ele é chamado de rede neural profunda (DNN). Existem casos específicos de DNNs, como mostrado na Figura 3.1 e explicados a seguir.

Rede Neural Totalmente Conectada (FCNN): A saída de cada camada da FCNN, Figura 3.1(a), ou Perceptron Multicamadas (MLP), é alimentada para a próxima camada. A saída de um neurônio é diretamente passada e ativada pelos neurônios da próxima camada (COLLOBERT; BENGIO, 2004). As FCNNs são usadas para extração de características e aproximação de funções, mas apresentam alta complexidade, desempenho modesto e convergência lenta.

Autoencoder (AE): AE, Figura 3.1(b), consiste em duas redes neurais que replicam a entrada para a saída em aprendizado não supervisionado. A primeira rede codifica as características da entrada, e a segunda decodifica essas características para restaurar a entrada original. AEs são usados para classificar e armazenar dados de alta dimensão devido à sua capacidade de aprender as características úteis de baixa dimensão dos dados

de entrada para recuperar os dados de entrada (MANNING; SCHUTZE, 1999).

Rede Neural Convolucional (CNN): Utilizando operações de pooling e filtros móveis, as CNNs, Figura 3.1(c), capturam correlações entre dados adjacentes, gerando níveis mais altos de abstração. Comparadas às FCNNs, as CNNs extraem características enquanto reduzem a complexidade do modelo e o risco de overfitting (ZEILER; FERGUS, 2014), sendo eficazes no processamento de imagens e dados estruturados.

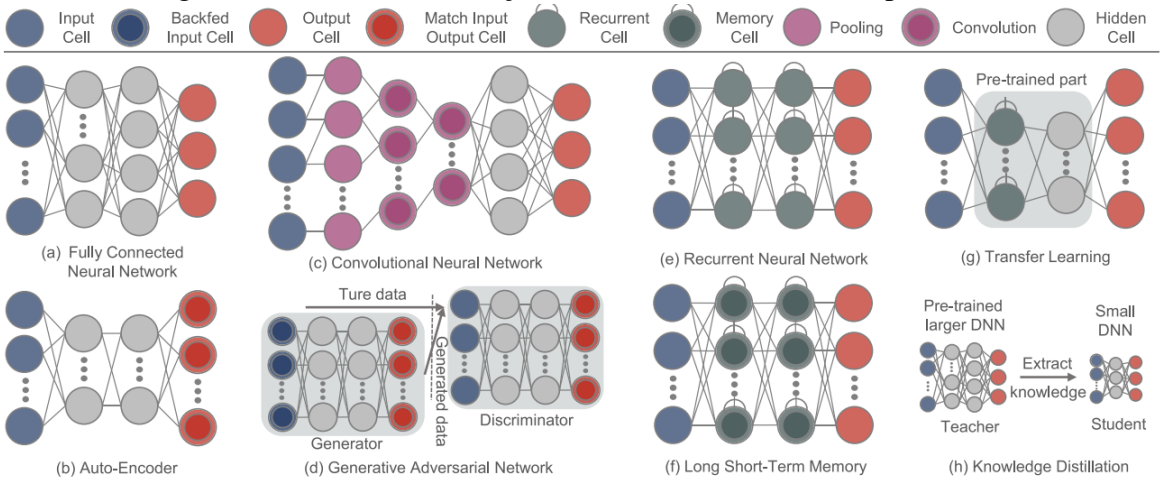
Rede Adversária Generativa (GAN): Composta por um gerador e um discriminador, a GAN, Figura 3.1(d), usa um processo adversarial para otimizar a geração e distinção de dados até alcançar um equilíbrio de Nash (GOODFELLOW et al., 2014).

Rede Neural Recorrente (RNN): As RNNs, Figura 3.1(e), lidam com dados sequenciais, recebendo informações da camada superior e do canal anterior. São ideais para prever informações futuras ou restaurar dados sequenciais ausentes. No entanto, sofrem com a explosão do gradiente, problema mitigado pelas LSTMs, Figura 3.1(f), que adicionam uma estrutura de porta e célula de memória para controlar o fluxo de informações (HOCHREITER; SCHMIDHUBER, 1997).

Transfer Learning (TL): TL, Figura 3.1(g), transfere conhecimento do domínio fonte para o alvo, melhorando o desempenho de aprendizado no domínio alvo. Recentemente, o Knowledge Distillation (KD) (HINTON; VINYALS; DEAN, 2015a) emergiu como uma forma de TL, onde conhecimento de um modelo grande (professor) é transferido para um modelo menor (aluno), como mostrado na Figura 3.1(h), permitindo que este último alcance um desempenho ótimo com menor complexidade.

Transformer: Os modelos Transformers, revolucionaram o campo do aprendizado profundo, especialmente em aplicações de Processamento de Linguagem Natural (PLN), como tradução automática e geração de texto, mas também expandiram sua utilização para outras áreas, como Visão Computacional. A arquitetura do Transformer baseia-se inteiramente em mecanismos de atenção, particularmente na autoatenção, permitindo que ele modele relações de longo alcance nos dados de forma eficiente (VASWANI et al., 2017). Sua ascensão deve-se ao desempenho superior em tarefas complexas, à capacidade de lidar com grandes volumes de dados e ao uso de aprendizado paralelo, que o torna significativamente mais rápido e escalável em comparação com arquiteturas tradicionais como as RNNs. Recentemente, modelos baseados em Transformers, como o BERT e o GPT, têm demonstrado desempenho de ponta em benchmarks de PLN, solidificando sua posição como uma das principais arquiteturas em IA.

Figura 3.1: Estruturas e funções básicas de DNNs e DL típicos



Fonte: (WANG et al., 2020)

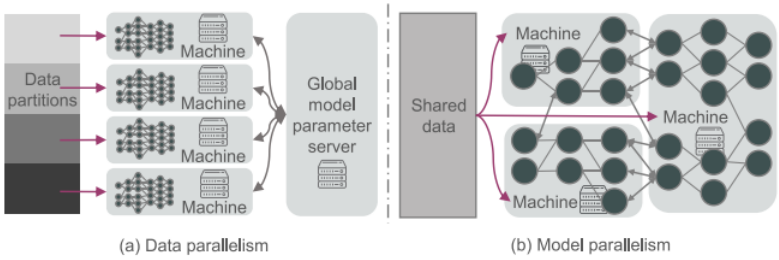
3.1.3 Treinamento de Aprendizado Profundo

No treinamento de aprendizado profundo, a computação ocorre na ordem inversa. Múltiplas iterações são realizadas através das camadas para otimizar os parâmetros de cada camada de multiplicações de matrizes, começando pela última camada e indo até a primeira. O algoritmo utilizado é geralmente o gradiente descendente estocástico. Em cada iteração, uma "pequena parcela"de amostras é selecionada aleatoriamente e usada para atualizar os gradientes na direção que minimiza a perda de treinamento. Uma passagem por todo o conjunto de dados de treinamento é chamada de época de treinamento (RUDER, 2016).

Uma forma comum de treinar modelos de DL é de maneira centralizada, porém, consome muito tempo e recursos. Como uma possível solução, temos o treinamento distribuído que pode acelerar o processo. Existem duas abordagens principais para o treinamento distribuído: paralelismo de dados e paralelismo de modelo, como mostrado na Figura 3.2. O paralelismo de modelo divide um grande modelo em várias partes para treinamento paralelo, melhorando a velocidade e lidando com limitações de memória, muitas vezes usando GPUs distribuídas (COATES et al., 2013). O paralelismo de dados divide os dados em partes, treinando cópias do modelo em paralelo com diferentes amostras de dados, aumentando a eficiência (MORITZ et al., 2015).

Por sorte, temos um grande número de dispositivos finais, nós de borda e data centers em nuvem que estão dispersos, e mais previstos para serem conectados por meio de redes de computação de borda. Todos esses dispositivos distribuídos podem potencial-

Figura 3.2: Treinamento distribuído em termos de paralelismo de dados e modelos



Fonte: (WANG et al., 2020)

mente ser contribuidores poderosos caso o treinamento de DL saia da nuvem.

3.1.4 Frameworks

Quando os pesquisadores desejam explorar modelos de aprendizado profundo, frequentemente se voltam para bibliotecas de software de código aberto e kits de desenvolvimento de hardware. Existem várias opções de bibliotecas de software de código aberto disponíveis para inferência e treinamento de aprendizado profundo em dispositivos finais e servidores de borda. No entanto, cada uma delas têm seus próprios cenários de aplicação. Para implantar o aprendizado profundo na borda, são necessárias bibliotecas eficientes e leves.

O **TensorFlow**, desenvolvido pelo Google e lançado em 2015, é uma dessas bibliotecas. Ele oferece uma interface para expressar algoritmos de aprendizado de máquina e uma implementação para executá-los em sistemas distribuídos heterogêneos. O fluxo de trabalho de computação do TensorFlow é representado como um gráfico direcionado e utiliza um algoritmo de alocação para distribuir tarefas de computação com base no tempo de execução estimado ou medido e no tempo de comunicação(ABADI et al., 2016a). O TensorFlow pode ser implantado em dispositivos de borda, como Raspberry Pi e smartphones. Apesar disso, introduzido no final de 2017, o **TensorFlow Lite**, uma versão otimizada para dispositivos móveis e embarcados, foi criado. Com suporte a GPU móvel adicionado em 2019, o TensorFlow Lite oferece apenas capacidades de inferência no dispositivo, sem suporte para treinamento, e alcança baixa latência comprimindo modelos DNN pré-treinados.

Outra estrutura de aprendizado profundo é o **Caffe**(JIA et al., 2014), originalmente desenvolvido por Jia e atualmente mantido pelo Facebook, com a versão atualizada chamada **Caffe2**. O Caffe2 busca fornecer uma maneira fácil e direta para o aprendizado

profundo, especialmente em dispositivos móveis, como smartphones e Raspberry Pis. O **PyTorch**(FACEBOOK,), também desenvolvido pelo Facebook, tem um objetivo principal diferente do Caffe2, com foco na integração de protótipos de pesquisa para o desenvolvimento de produção. Entretanto, o Facebook realizou a fusão do Caffe2 e do PyTorch inteiramente no PyTorch.

As GPUs são essenciais para a eficiência tanto na inferência quanto no treinamento de modelos DNN. A Nvidia oferece bibliotecas de software como **CUDA**(NVIDIA, a) e **cuDNN**(NVIDIA, b), voltadas para o processamento de GPU em aprendizado profundo. Embora sejam amplamente utilizadas em servidores desktop para treinamento de modelos, não estão tão disponíveis em dispositivos móveis. Para experimentar com dispositivos de borda, como o Nvidia Jetson TX2, os pesquisadores podem usar kits de desenvolvimento específicos e SDKs fornecidos pela Nvidia. Outra opção popular é o kit Intel Edison, especialmente projetado para experimentos de IoT.

Outras bibliotecas para aprendizado profundo, além das já citadas, estão presentes na Figura 3.3, onde são destacados os suportes respectivos.

3.2 Computação de Borda

A edge computing tornou-se uma solução crucial para superar os obstáculos das tecnologias emergentes, graças às suas vantagens em reduzir a transmissão de dados, melhorar a latência dos serviços e aliviar a carga sobre a computação em nuvem. A arquitetura de edge computing será um complemento significativo para a nuvem, podendo até substituir sua função em certos contextos(BILAL et al., 2018) (MOURADIAN et al., 2017).

As definições e divisões dos dispositivos de borda são muitas vezes ambíguas na literatura. Por isso, é útil dividir os dispositivos de borda em dispositivos finais (como smartphones e veículos inteligentes) e nós de borda (como Cloudlets, nós de névoa, servidores de borda e servidores MEC). Dispositivos finais referem-se aos móveis e de IoT, enquanto os nós de borda incluem servidores implantados na borda da rede. Como pode ser visto na Figura 3.4.

O termo "computação de borda"foi dado ao conjunto de novas tecnologias que surgiram destinadas a operar na borda de rede, com os mesmos princípios, mas com focos diferentes. No entanto, a comunidade de edge computing ainda não chegou a um acordo sobre definições padronizadas, arquiteturas e protocolos para a computação de

Figura 3.3: Bibliotecas de aprendizado profundo e seus suportes respectivos.

POTENTIAL DL LIBRARIES FOR EDGE COMPUTING

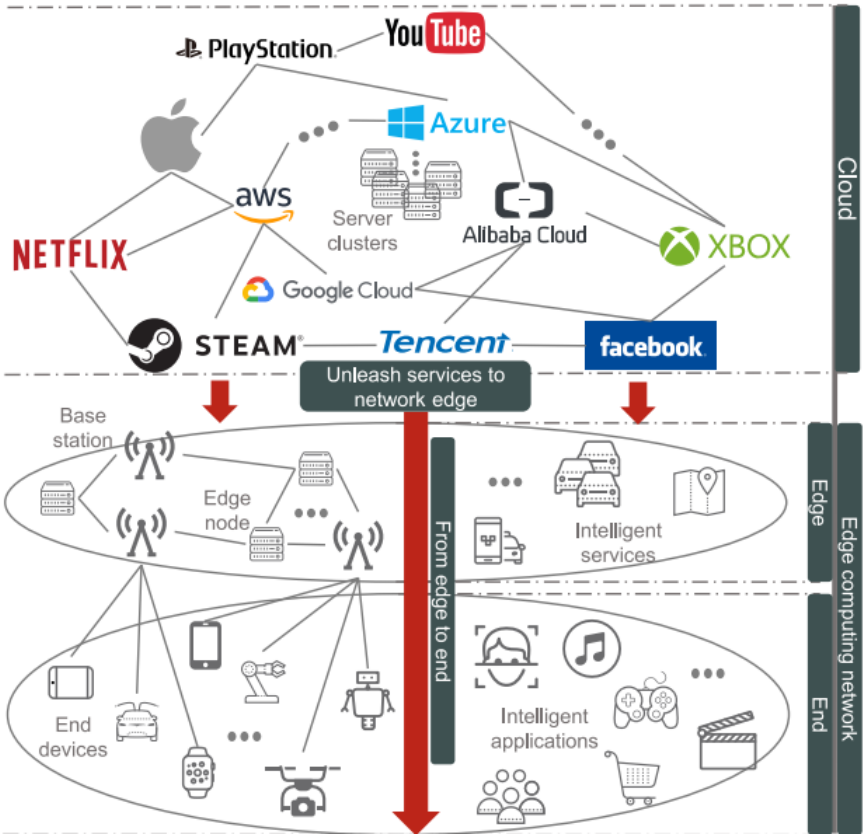
Library	CNTK [56]	Chainer [57]	TensorFlow [58]	DL4J [59]	TensorFlow Lite [60]	MXNet [61]	(Py)Torch [62]	CoreML [63]	SNPE [53]	NCNN [64]	MNN [65]	Paddle-Mobile [66]	MACE [67]	FANN [68]
Owner	Microsoft	Preferred Networks	Google	Skymind	Google	Apache Incubator	Facebook	Apple	Qualcomm	Tencent	Alibaba	Baidu	XiaoMi	ETH Zürich
Edge Support	×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Android	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	×
iOS	×	×	×	×	×	✓	✓	✓	×	✓	✓	✓	✓	×
Arm	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
FPGA	×	×	×	×	×	×	✓	×	×	×	×	✓	×	×
DSP	×	×	×	×	×	×	×	×	✓	×	×	×	×	×
GPU	✓	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×	×
Mobile GPU	×	×	×	×	✓	×	×	✓	✓	✓	✓	✓	✓	×
Training Support	✓	✓	✓	✓	×	✓	✓	×	×	×	×	×	×	✓

Fonte: (WANG et al., 2020)

borda (BILAL et al., 2018). Algumas dessas tecnologias são descritas a seguir:

- *Cloudlet e Micro Data Centers*: Cloudlets são elementos de arquitetura de rede que combinam computação móvel e computação em nuvem, representando a camada intermediária na arquitetura de três camadas: dispositivos móveis, micro nuvem e nuvem. Seus destaques incluem a definição do sistema e a criação de algoritmos para suportar computação em nuvem de borda de baixa latência, além da implementação de funcionalidades em código aberto como uma extensão do software OpenStack(SATYANARAYANAN et al., 2009). Semelhantes aos Cloudlets, os MDCs(AAZAM; HUH, 2015) são projetados para complementar a nuvem, embalando todo o equipamento necessário para computação, armazenamento e rede em um único invólucro. Isso cria um ambiente seguro e autônomo para aplicações que necessitam de menor latência ou para dispositivos finais com baterias de vida

Figura 3.4: Exemplo Cloud-edge-end



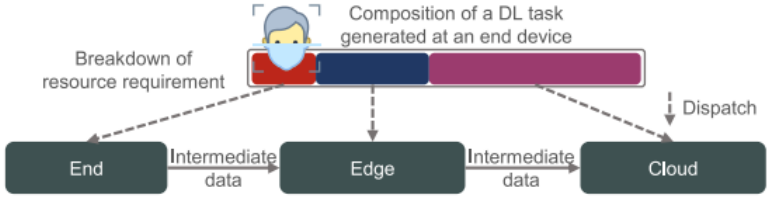
Fonte: (WANG et al., 2020)

útil limitada.

- *Fog Computing*: A Fog Computing adota uma arquitetura de nuvem distribuída em várias camadas, abrangendo bilhões de dispositivos e grandes centros de dados em nuvem(BONOMI et al., 2012). Embora compartilhe serviços semelhantes com a nuvem, a névoa é implantada em áreas geográficas específicas e é ideal para aplicações que exigem resposta em tempo real com baixa latência, como IoT e aplicações interativas. Diferente dos Cloudlets, MDCs e MEC, a computação em névoa é mais focada em dispositivos IoT.
- *Computação de Borda Móvel (Multi-acesso) (MEC)*: A Computação de Borda Móvel coloca capacidades de computação e ambientes de serviço na borda das redes celulares(PATEL et al., 2014), proporcionando menor latência, consciência de contexto, localização e maior largura de banda. A implantação de servidores de borda nas Estações Base (BSs) celulares permite que os usuários implantem novas aplicações e serviços de forma flexível e rápida. O ETSI ampliou a terminologia de MEC para Computação de Borda Multi-acesso, acomodando tecnologias como Wi-Fi.
- *Computação Colaborativa Fim-Borda-Nuvem*: A computação colaborativa fim-

borda-nuvem, como mostrado na Figura 3.5, é um novo paradigma onde tarefas de menor intensidade computacional são executadas nos dispositivos finais ou descarregadas para a borda, evitando o atraso de enviar dados para a nuvem. Tarefas intensivas em computação são segmentadas e despachadas entre o fim, borda e nuvem, reduzindo o atraso de execução e assegurando a precisão dos resultados(KANG et al., 2017), (LI et al., 2018). Este paradigma busca não apenas concluir as tarefas com sucesso, mas também alcançar um equilíbrio ótimo entre consumo de energia, carga dos servidores e atrasos de transmissão e execução. Por estes motivos ela será o foco dos testes realizados neste artigo.

Figura 3.5: Esboço de computação DL colaborativa em nuvem de ponta



Fonte: (WANG et al., 2020)

3.2.1 Harware

Para possibilitar a utilização da AI tanto na borda quanto em quaisquer outros dispositivos, devemos fornecer algum hardware com a capacidade necessária para essa tarefa tão custosa e específica.

O hardware de IA emergente para a borda pode ser classificado de acordo com sua arquitetura técnica em três principais categorias:

- *Unidades de Processamento Gráfico (GPU)*: As GPUs, como as da NVIDIA baseadas nas arquiteturas Ampere e Hopper, são conhecidas por sua compatibilidade e desempenho, mas tendem a consumir mais energia. A arquitetura Ampere, com Tensor Cores de terceira geração, e a Hopper, com o Transformer Engine e memória HBM3, oferecem melhorias significativas em eficiência e desempenho para aplicações de IA(DAVIDSON, 2024).
- *Field Programmmable Gate Array (FPGA)*: Os FPGAs (JIANG et al., 2018c) economizam energia e requerem menos recursos de computação em comparação com as GPUs, mas têm menor compatibilidade e capacidade de programação limitada.

- *Circuito Integrado de Aplicação Específica (ASIC)*: Este hardware, como o TPU do Google e a série Ascend da HiSilicon, é projetado de forma personalizada para oferecer desempenho estável e eficiente em termos de consumo de energia

Os smartphones, sendo os dispositivos de borda mais amplamente implantados, têm visto um rápido desenvolvimento nos chips para IA. A Qualcomm lançou recentemente o Snapdragon 8 Gen 3 e o Snapdragon 8s Gen 3, que incluem NPUs para processamento local de IA. Esses chips permitem que aplicativos de IA generativa operem diretamente no dispositivo, melhorando a segurança dos dados e a eficiência de processamento. A Qualcomm também introduziu o Qualcomm AI Hub, que oferece uma biblioteca de modelos de IA pré-otimizados para facilitar a implementação em dispositivos Snapdragon(Qualcomm, 2024b) (Qualcomm, 2024a). A MediaTek lançou recentemente o Dimensity 9300, que inclui a sétima geração de processadores de IA e uma APU para acelerar a computação de redes neurais e processamento generativo de IA no dispositivo. Este chip usa um design de núcleo "All Big Core"que prioriza o desempenho bruto, destacando-se pela eficiência e capacidade de lidar com tarefas de IA avançadas(YUGATECH, 2024).

Avanços recentes incluem a previsão de chips de IA de próxima geração mais eficientes em termos de energia e personalizados para aplicações multimodais e modelos de linguagem, bem como arquiteturas híbridas de borda-nuvem que melhoram a velocidade e reduzem a latência(EdgeCortix, 2024). A CES 2024 apresentou inovações da Intel, AMD e Nvidia, como os processadores Raptor Lake Refresh da Intel, os chips Ryzen 8000G da AMD com NPUs integradas, e as GPUs GeForce RTX Super da Nvidia, projetadas tanto para jogos quanto para cargas de trabalho de IA(DAVIDSON, 2024).

3.2.2 Framework Para Computação Distribuída

Juntamente com o desenvolvimento de hardware, temos o desenvolvimento de frameworks que estão surgindo como soluções para sistemas de computação de borda. Para serviços que demandam configurações complexas e grande uso de recursos, os sistemas de computação de borda com arquitetura de microsserviços avançada são considerados a direção futura do desenvolvimento. Atualmente, o Kubernetes é amplamente reconhecido como o principal sistema de orquestração de contêineres para implantação, gerenciamento e escalonamento de aplicativos na computação em nuvem (BERNSTEIN, 2014). A Huawei desenvolveu sua solução de computação de borda chamada "KubeEdge", construída

sobre o Kubernetes, para gerenciar rede, implantação de aplicativos e sincronização de metadados entre a nuvem e a borda (também integrado ao Akraino Edge Stack)(XIONG et al., 2018). O "OpenEdge"foca em garantir a segurança do framework de computação e simplificar o desenvolvimento de aplicativos (Baidu, 2019). Para a Internet das Coisas (IoT), o Azure IoT Edge e o EdgeX foram projetados para levar inteligência da nuvem para a borda, permitindo a implantação e execução de inteligência artificial em uma variedade de dispositivos IoT (Microsoft, 2019).

Abaixo, na Tabela 3.2 consta os dados mais atualizados de produtos fornecidos por grandes empresas como soluções de hardware e frameworks para IA na borda.

Tabela 3.2: Comparação de produtos e recursos de Edge Computing

Owner	**Production**	**Feature**
Integrated Commodities		
Microsoft	Data Box Edge	Data preprocessing and transmission
Intel	Movidius Neural Compute Stick	Platform-agnostic, plug-and-play
NVIDIA	Jetson	Low power consumption (5W)
Huawei	Atlas Series	Device, edge, cloud integration
Qualcomm	Snapdragon 8 Series	Adaptability to DL frameworks
HiSilicon	Kirin 600/900 Series	Independent NPU
AI Hardware for Edge Computing		
HiSilicon	Ascend Series	Low to high energy consumption scenarios
MediaTek	Helio P60	Accelerates neural network computing
NVIDIA	Ampere GPUs	High capabilities, high energy consumption
Google	TPU	Stable performance and power consumption
Intel	Xeon D-2100	Power- and space-efficient
Samsung	Exynos 9820	Mobile AI tasks acceleration
Huawei	KubeEdge	Edge-cloud collaboration
Baidu	OpenEdge	Simplifies application production
Edge Computing Frameworks		
Microsoft	Azure IoT Edge	Remote edge management
Linux Foundation	EdgeX	Industrial and enterprise IoT edge
Linux Foundation	Akraino Edge Stack	Distributed cloud edge platform
NVIDIA	NVIDIA EGX	Real-time edge processing
Amazon	AWS IoT Greengrass	Intermittent connectivity support
Google	Google Cloud IoT	Integration with Google AI products

Fonte: Os Autores

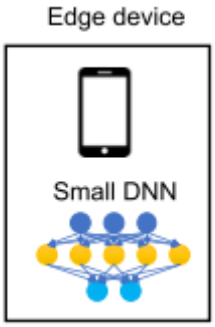
4 INFERÊNCIA

Para que as aplicações mencionadas alcancem suas metas de latência, várias estratégias de offloading foram propostas para melhorar o processo de inferência em Redes Neurais Profundas (DNN). Neste capítulo, analisaremos três principais locais de processamento: 1) computação local no dispositivo final, que se refere a dispositivos como smartphones, dispositivos IoT e outros aparelhos móveis ou embarcados; 2) arquiteturas baseadas em servidores de borda, onde os dados são enviados para servidores localizados em locais próximos ao usuário; e 3) computação conjunta entre dispositivos finais, servidores de borda e a nuvem, aproveitando a colaboração entre diferentes camadas de infraestrutura para otimizar a latência e o consumo de recursos.

4.1 Modelos

Pesquisadores têm focado em maneiras de reduzir a latência do aprendizado profundo em dispositivos com recursos limitados, como o da Figura 4.1. Esses esforços beneficiam o ecossistema de borda, melhorando o desempenho das Redes Neurais Profundas (DNN) em dispositivos finais ou servidores de borda.

Figura 4.1: Computação no Dispositivo Final



Fonte: (CHEN; RAN, 2019)

4.1.1 Design de Modelos

Ao projetar modelos DNN para dispositivos com recursos limitados, os pesquisadores visam reduzir o número de parâmetros, diminuindo assim a memória e a latência de execução, sem comprometer a precisão. Algumas técnicas e modelos populares estão dis-

poníveis em plataformas de aprendizado de máquina de código aberto, como Tensorflow, para inicialização rápida e incluem MobileNets (HOWARD et al., 2017), decompõem filtros de convolução em operações mais simples, reduzindo o número de cálculos necessários; SSD (Single Shot Multibox Detector) (LIU et al., 2016) e YOLO (REDMON; FARHADI, 2017), detectores de disparo único que preveem simultaneamente a localização e a classe do objeto, acelerando o processamento; e SqueezeNet (IANDOLA et al., 2016), Usa filtros de convolução de 1x1 para diminuir a quantidade de dados processados.

4.1.2 Compressão de Modelos

A compressão de modelos DNN é essencial para viabilizar o uso de redes neurais profundas em dispositivos de borda, buscando reduzir o tamanho dos modelos sem comprometer significativamente sua precisão. As principais técnicas de compressão incluem quantização de parâmetros, poda de parâmetros e destilação de conhecimento.

A quantização de parâmetros transforma os números de ponto flutuante dos modelos em números de menor largura de bits, o que reduz a necessidade de operações complexas de ponto flutuante. Já a poda remove parâmetros menos significativos, como aqueles próximos de zero, diminuindo a complexidade do modelo. Ambas as técnicas já foram e podem ser aplicadas separadamente ou em conjunto (HAN; MAO; DALLY, 2015). Exemplos notáveis incluem DeepIoT(YAO et al., 2017), que utiliza poda em estruturas comuns de aprendizado profundo para dispositivos IoT, e CMSIS-NN(LAI; SUDA, 2018), uma biblioteca para processadores ARM Cortex-M que maximiza a performance através da quantização e otimização da multiplicação de matrizes. Outro exemplo notável foi realizado em (HAN et al., 2017) no qual mostraram um ganho de 10x em aceleração por poda e 2x por quantização em modelos RNN. A destilação de conhecimento (HINTON; VINYALS; DEAN, 2015b) cria uma DNN menor que imita o comportamento de uma maior, treinando a rede menor com as saídas da maior. Outra técnica, chamada saída rápida (TEERAPITTAYANON; MCDANEL; KUNG, 2016), calcula apenas algumas camadas iniciais para fornecer classificações aproximadas.

Combinações dessas técnicas têm sido exploradas em outros estudos. Adadeep (LIU et al., 2018) escolhe automaticamente entre diferentes técnicas de compressão para atender às necessidades específicas de aplicações móveis, enquanto DeepMon (HUYNH; LEE; BALAN, 2017) combina quantização com cache de resultados de camadas intermediárias em GPUs, reutilizando resultados de computação de quadros anteriores para

reduzir cálculos redundantes e acelerar a execução.

4.2 Harware

Para acelerar a inferência em aprendizado profundo, fabricantes de hardware utilizam CPUs e GPUs existentes, além de desenvolver ASICs personalizados, como a unidade de processamento tensorial (TPU) do Google (Google Cloud,) . ShiDianNao (DU et al., 2015) é outro ASIC personalizado que foca em acessos eficientes à memória para reduzir latência e consumo de energia, sendo parte da família de aceleradores DNN DianNao (CHEN et al., 2016), voltada para dispositivos embarcados . Aceleradores DNN baseados em FPGA também são promissores, oferecendo computação rápida e reconfigurabilidade . Esses designs de ASICs e FPGA são geralmente mais eficientes em termos de energia comparados às CPUs e GPUs tradicionais, que consomem mais energia por suportar diversas cargas de trabalho.

Os fornecedores de hardware também fornecem ferramentas de software para os desenvolvedores de aplicativos aproveitarem as acelerações fornecidas pelo hardware. Os fabricantes de chips desenvolveram ferramentas de software para otimizar o aprendizado profundo nos chips existentes, como o Intel OpenVINO Toolkit. A versão mais recente, OpenVINO 2024.2, traz melhorias contínuas em desempenho, especialmente para cargas de trabalho de IA generativa. Essa atualização oferece otimizações para CPUs, GPUs integradas e discretas, além de suporte expandido para modelos de linguagem grande (LLM) e métodos de compressão de modelos. A Nvidia, por exemplo, recentemente atualizou sua plataforma EGX para oferecer suporte a uma gama mais ampla de dispositivos, incluindo servidores T4 e os novos A30 e A100. A Qualcomm, por sua vez, lançou o AI Hub em 2024, oferecendo uma biblioteca de mais de 75 modelos de IA otimizados para rodar em diversas plataformas Qualcomm, incluindo dispositivos móveis, VR/AR, automotivos e PCs. Este hub promete reduzir o tempo de lançamento no mercado ao simplificar a integração de modelos de IA nos processadores Snapdragon, usando o SDK de IA da Qualcomm

Abordagens de software também ajudam a utilizar eficientemente o hardware, como o trabalho realizado em (LANE et al., 2016), que decompõem DNNs e as distribuem entre processadores heterogêneos, como CPU e GPU, para acelerar a execução.

Apesar das técnicas de aceleração de hardware e compressão ajudarem na execução de DNNs em dispositivos finais, ainda é muito desafiador implantar DNNs grandes

e exigentes em tempo real nesses dispositivos devido as limitações de recursos. Naturalmente, como alternativa se considera a transferência das computações das DNNs para entidades mais robustas, como servidores de borda ou a nuvem. Contudo, a nuvem não é ideal para aplicações de borda que requerem respostas rápidas, como mostrado em (SHI et al., 2016). Sendo assim, o servidor de borda, que está mais próximo dos usuários e pode responder rapidamente às solicitações, torna-se a escolha ideal. Uma abordagem direta é transferir toda a computação dos dispositivos finais para os servidores de borda. Nesse contexto, os dispositivos finais enviam seus dados para um servidor de borda próximo e recebem os resultados processados pelo servidor. Por exemplo, em (WANG; WANG; MAO, 2018) sempre transferiram as DNNs para o servidor de borda (um gateway IoT) para analisar sinais sem fio.

4.2.1 Gerenciamento de Recursos

Executar DNNs em servidores de borda requer gerenciar eficientemente as tarefas de vários dispositivos finais em recursos computacionais compartilhados. Diversos trabalhos focam justamente nos trade-offs entre precisão, latência e outros méritos de desempenho, como o número de solicitações atendidas. O VideoStorm (ZHANG et al., 2017) foi pioneiro ao esboçar esses trade-offs para selecionar a configuração adequada da DNN para cada solicitação, atendendo aos objetivos de precisão e latência. As configurações também podem ser atualizadas online durante o streaming de vídeo, como feito no Chameleon (JIANG et al., 2018b). Em Mainstream (JIANG et al., 2018a), são abordados os trade-offs entre precisão e latência em servidores de borda usando aprendizado por transferência para reduzir recursos computacionais, permitindo que várias aplicações compartilhem camadas inferiores comuns de DNN e computem camadas superiores específicas para cada aplicação, diminuindo a computação total necessária.

4.3 Execução no Contínuo Móvel-Borda-Nuvem

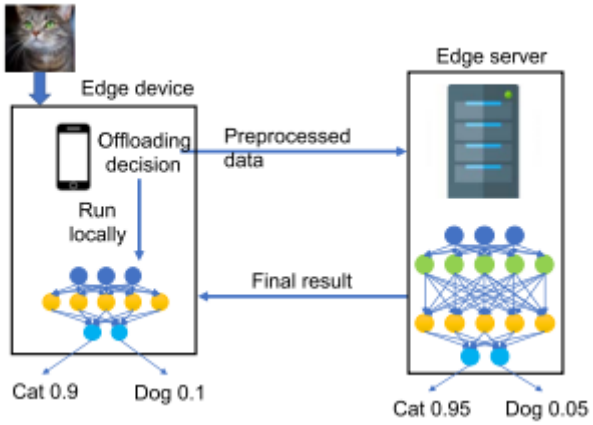
Embora o servidor de borda e a nuvem possam acelerar o processamento DNNs, nem sempre é necessário que os dispositivos de borda executem essas redes diretamente nos servidores de borda, podemos recorrer ao descarregamento inteligente. Existem quatro cenários principais de descarregamento:

- *Descarregamento Binário*: descarregamos completamente ou não descarregamos toda a DNN.
- *Descarregamento Parcial*: é decidido qual parte das computações da DNN deve ser descarregada.
- *Arquiteturas Hierárquicas*: a combinação entre dispositivos de borda, servidores de borda e a nuvem.
- *Computação distribuída*: onde a DNN é processada entre vários dispositivos que possuem a mesma hierarquia.

4.3.1 Offloading

Abordagens como as realizadas em DeepDecision (RAN et al., 2018) e MCDNN (HAN et al., 2016) utilizaram de estratégias de descarregamento baseadas em otimização, levando em conta restrições como latência de rede, largura de banda, energia do dispositivo e custo monetário. Essas decisões são fundamentadas em medições observadas em trade-offs entre energia, precisão, latência e tamanho de entrada para diferentes modelos de DNN. As opções de DNN disponíveis podem incluir tanto modelos populares já existentes quanto novas variantes construídas por meio de destilação de conhecimento ou combinação de camadas de diferentes modelos de DNN. Para exemplificar um descarregamento combinado com seleção de modelo, onde uma DNN poderosa está no servidor de borda e uma DNN mais simples está no dispositivo final, temos a Figura 4.2 a seguir.

Figura 4.2: Descarregamento com seleção de modelo



Fonte: (CHEN; RAN, 2019)

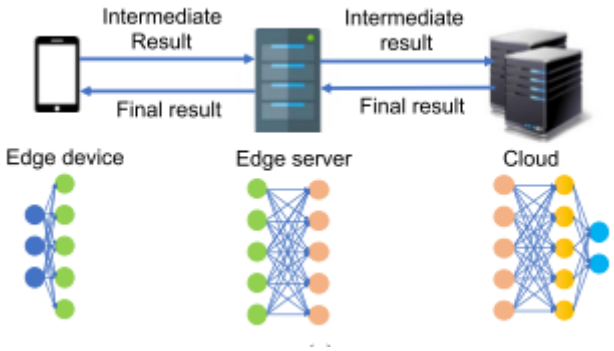
Apesar de o descarregamento ser amplamente estudado em redes e computação

de borda, o descarregamento de DNNs adiciona uma camada extra de complexidade, envolvendo decisões sobre onde e qual modelo ou parte do modelo executar. Essas decisões dependem de fatores como tamanho dos dados, capacidades de hardware, modelo de DNN a ser usado e qualidade da rede.

4.3.2 Particionamento de Modelos

O particionamento de modelos de DNN pode utilizar uma abordagem de descarregamento fracionário, aproveitando a estrutura das camadas das DNNs. Nesse método, algumas camadas são processadas no dispositivo local e outras pelo servidor de borda ou na nuvem. Essa estratégia busca reduzir a latência utilizando os recursos computacionais de outros dispositivos de borda. No entanto, é crucial garantir que a latência de comunicação dos resultados intermediários não elimine os benefícios obtidos. Um exemplar deste modelo pode ser visto a seguir na Figura 4.3.

Figura 4.3: Computação em dispositivos de borda com particionamento de modelo DNN



Fonte: (CHEN; RAN, 2019)

A lógica por trás do particionamento de modelos é que, após o processamento das primeiras camadas da DNN, o tamanho dos resultados intermediários é menor, facilitando a transmissão pela rede para um servidor de borda, ao invés de enviar os dados brutos originais (ZHAO; BARIJOUGH; GERSTLAUER, 2018). Uma ótima exemplificação é o Neurosurgeon (KANG et al., 2017), ferramenta que decide, camada por camada, de maneira inteligente onde particionar a DNN considerando as condições da rede. Além disso, também é possível dividir ao longo da dimensão de entrada, como selecionando apenas linhas específicas de uma imagem. Esse método permite um particionamento mais granular, crucial para dispositivos leves, como sensores IoT, que não têm capacidade de memória para armazenar uma camada inteira da DNN. No entanto, isso pode aumentar a

dependência dos dados, pois as camadas subsequentes requerem resultados das partições adjacentes.

Como exemplos de particionamento por entrada temos MoDNN (MAO et al., 2017) e DeepThings (ZHAO; BARIJOUGH; GERSTLAUER, 2018).Ambos abordam o descarregamento parcial em DNNs de maneira semelhante às técnicas de descarregamento não DNN que dividem uma aplicação em sub tarefas e decidem onde executá-las com base em considerações de energia e latência. No contexto do aprendizado profundo, a nova decisão envolve como dividir as sub tarefas, podendo ser camada por camada, por entrada ou outras formas ainda a serem exploradas.

4.3.3 Borda e Nuvem

Como mostrado na Figura 4.3, computação de aprendizado profundo pode ser realizada tanto em dispositivos de borda quanto na nuvem. Utilizar apenas a nuvem pode comprometer requisitos de tempo real, mas aproveitar os recursos da nuvem pode diminuir o tempo de processamento. Em vez de escolher entre borda ou nuvem, algumas abordagens particionam redes neurais profundas (DNNs), executando diferentes camadas em diferentes locais. No trabalho realizado em (LI; OTA; DONG, 2018), um modelo de DNN foi dividido entre o servidor de borda e a nuvem: o servidor de borda processa as camadas iniciais e a nuvem processa as camadas superiores, enviando os resultados de volta para os dispositivos finais. Nesse método, a nuvem pode ajudar com solicitações computacionalmente pesadas e aumentar a taxa de processamento de solicitações do servidor de borda enquanto reduz o tráfego de rede entre o servidor de borda e a nuvem.

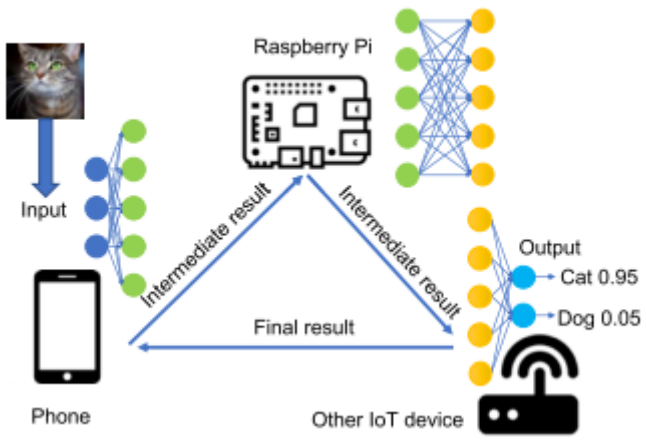
No caso de DDNN (TEERAPITTAYANON; MCDANEL; KUNG, 2017), a computação é distribuída entre a nuvem, servidores de borda e dispositivos finais, permitindo que algumas solicitações sejam resolvidas localmente, por meio de saída rápida, para evitar atrasos. Isso pode ser realizado, pois uma característica única da borda é que o servidor de borda serve geralmente a usuários de uma determinada região, sugerindo que seus dados de entrada e, consequentemente, suas saídas de DNN podem ter semelhança. Já em Precog (DROLIA; GUO; NARASIMHAN, 2017), utiliza-se de modelos menores e especializados em dispositivos finais com base em dados recentes. Se a classificação falhar, a consulta é enviada ao servidor de borda, que armazena modelos completos, combinando eficácia e eficiência. Apesar da avaliação não ser feita por DNNs, assemelha-se a destilação de conhecimento para modelos compactados na qual usa uma combinação de

modelos de classificação mais fracos e mais fortes.

4.3.4 Distribuição de Carga

Além do descarregamento de computação para servidores de borda ou nuvem, há abordagens que distribuem a computação das DNNs entre vários dispositivos de borda auxiliares, mostrado na Figura 4.4. Exemplos destas abordagens são novamente os trabalhos realizados em MoDNN (MAO et al., 2017) e DeepThings (ZHAO; BARIJOUGH; GERSTLAUER, 2018), que dividem a execução das DNNs em dispositivos leves como Raspberry Pi e smartphones Android, com base nas capacidades de computação e memória desses dispositivos. Durante a execução, os dados de entrada são distribuídos entre os dispositivos auxiliares, cada um seguindo princípios de balanceamento de carga próprios. Em MoDNN usa-se um modelo similar ao MapReduce, enquanto em DeepThings utiliza-se uma heurística específica. A alocação de dados pode ser ajustada em tempo real para refletir mudanças na disponibilidade de recursos ou nas condições da rede. Também podem ser aplicados mecanismos formais de sistemas distribuídos para garantir desempenho consistente.

Figura 4.4: Computação distribuída com particionamento de modelo DNN



Fonte: (CHEN; RAN, 2019)

4.3.5 Preprocessamento

O preprocessamento de dados ao enviar informações para um servidor de borda pode reduzir a redundância deles e diminuir o tempo de comunicação. O Glimpse (CHEN

et al., 2015), por exemplo, transfere toda a computação de DNN para um servidor de borda próximo, mas filtrando quais quadros da câmera devem ser enviados utilizando a detecção de mudanças. Caso nenhuma mudança seja detectada, o Glimpse fará o rastreio dos quadros localmente no dispositivo final, melhorando o processamento e possibilitando o reconhecimento de objetos em tempo real em dispositivos móveis. Seguindo a mesma linha de raciocínio, em (LIU et al., 2017), foi construído um sistema de reconhecimento de alimentos que utiliza duas etapas de preprocessamento: na primeira etapa, são descartadas as imagens borradas, enquanto na segunda, recortam a imagem para incluir apenas os objetos de interesse. Essas etapas, apesar de leves, podem reduzir significativamente a quantidade de dados transferidos.

5 TREINAMENTO

Este capítulo aborda o treinamento de modelos de redes neurais, com ênfase em sua aplicação no contexto móvel-borda-nuvem. Entretanto, o foco principal não recai sobre o treinamento durante os testes, visto que o objetivo central é analisar os tempos de execução das inferências. Isso se deve ao fato de que, tradicionalmente, o treinamento de modelos é realizado de maneira offline, em servidores centralizados, utilizando grandes volumes de dados. Após o treinamento, o modelo treinado é distribuído para dispositivos de borda, onde realiza-se o processo de inferência. Embora essa abordagem seja amplamente adotada, ela enfrenta desafios significativos, especialmente no que diz respeito à privacidade dos dados e ao elevado consumo de largura de banda.

Para mitigar essas limitações, o treinamento na borda emerge como uma solução, onde o treinamento é realizado diretamente nos dispositivos finais. Isso reduz a necessidade de transmissão de dados para a nuvem, protegendo a privacidade e otimizando os recursos computacionais. O treinamento na borda adapta conceitos de treinamento distribuído de DNNs (Deep Neural Networks) em centros de dados, como o paralelismo de dados, onde cada dispositivo (ou trabalhador) calcula gradientes localmente e envia atualizações para um servidor central, o nó de borda.

5.1 Treinamento na Borda

Um exemplo de treinamento na borda é o DeepCham (LI et al., 2016), que utiliza um servidor de borda como mestre para treinar modelos de reconhecimento de objetos, permitindo que dispositivos conectados compartilhem domínios semelhantes. O desempenho desse tipo de treinamento é influenciado por fatores como latência de comunicação, largura de banda e a capacidade computacional dos dispositivos de borda.

Além do treinamento na borda, existem outros tipos de treinamento de redes neurais importantes no contexto de modelos distribuídos e paralelizados:

- **Treinamento Centralizado**: Neste método, o treinamento é realizado em servidores centralizados que possuem grandes capacidades computacionais. Embora o treinamento centralizado seja eficiente em termos de velocidade, ele apresenta limitações em termos de privacidade e uso de largura de banda.
- **Treinamento Federado**: O treinamento federado permite que dispositivos finais

(como smartphones) treinem modelos localmente e compartilhem apenas as atualizações dos parâmetros, sem precisar enviar os dados brutos para a nuvem. Isso preserva a privacidade dos dados, mas a sincronização entre dispositivos pode aumentar a complexidade do processo.

- **Treinamento Paralelizado de Modelo**: Nesse método, diferentes partes do modelo são treinadas em paralelo, cada uma em diferentes dispositivos ou servidores. Isso acelera o treinamento, mas exige uma comunicação eficiente entre as partes do modelo.
- **Treinamento Híbrido**: Combina o treinamento local e centralizado, aproveitando tanto a computação local nos dispositivos de borda quanto o poder de servidores centralizados. Esse tipo de abordagem pode ser útil quando a carga de trabalho é muito alta para ser completamente gerida nos dispositivos de borda.

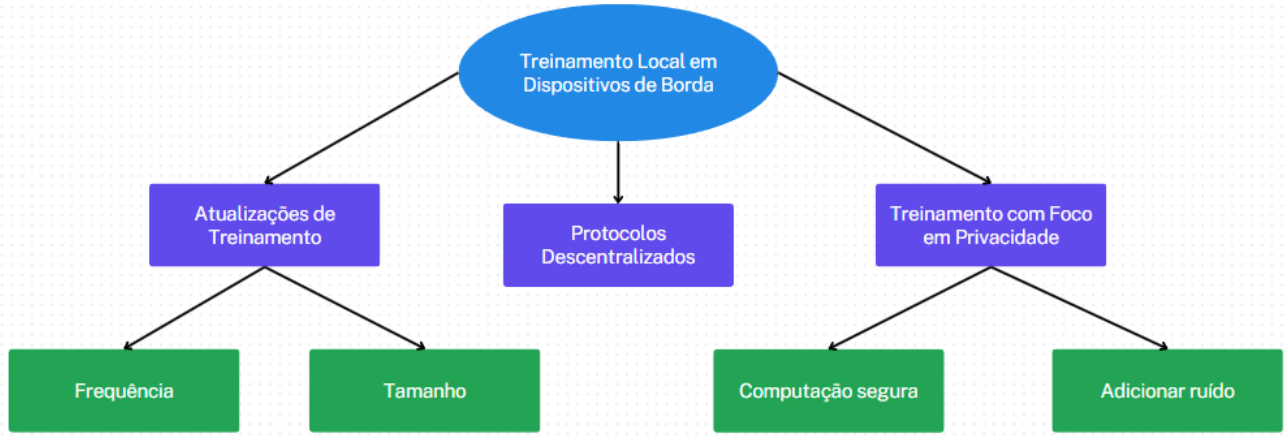
5.2 Técnicas de Treinamento na Borda

O treinamento distribuído em dispositivos de borda envolve diversas técnicas, que serão exploradas nas próximas seções. Algumas das principais técnicas incluem:

- **Frequência e Tamanho das Atualizações de Treinamento**: Define como e com que frequência as atualizações dos parâmetros são enviadas entre os dispositivos de borda e o servidor central.
- **Compartilhamento Descentralizado de Informações**: Dispositivos de borda compartilham informações de maneira descentralizada para melhorar a eficiência e reduzir a sobrecarga de comunicação.
- **Treinamento Preservando a Privacidade**: Técnicas como criptografia homomórfica e aprendizado federado permitem que os dados sejam processados localmente, preservando a privacidade.

A Figura 5.1 ilustra uma taxonomia dessas técnicas.

Figura 5.1: Taxonomia Treinamento



Fonte: Autores

5.3 Frequência de Atualizações

Os custos de comunicação são uma preocupação central nos dispositivos de borda, e reduzir a frequência e o tamanho das comunicações é essencial. Existem dois métodos principais para sincronizar as atualizações com um servidor central: a descida de gradiente estocástica (SGD) síncrona e assíncrona. Na SGD síncrona, as atualizações são sincronizadas após o cálculo dos gradientes, o que tende a alcançar melhores soluções, mas é mais lenta devido à espera pelos dispositivos mais lentos. Já na SGD assíncrona, as atualizações ocorrem de forma independente, sendo mais rápida, mas com o risco de usar informações desatualizadas, o que pode prejudicar a convergência. Métodos de treinamento distribuído buscam acelerar a SGD síncrona ou melhorar a convergência da SGD assíncrona. Utilizando a média elástica, é possível reduzir os custos de comunicação, permitindo mais cálculos locais antes da sincronização das atualizações. O aprendizado federado, por sua vez, lida com dados não independentes e não identicamente distribuídos, considerando cenários não ideais. Embora computar mais localmente economize custos de comunicação, pode resultar em *overfitting*, comprometendo a precisão. A política de controle proposta em (WANG et al., 2018) busca balancear a computação local e as atualizações globais, testada com Raspberry Pi e laptop.

Em (HSIEH et al., 2017), consideraram-se arquiteturas em camadas e seus custos de comunicação. É realizada a análise da SGD síncrona em dispositivos geograficamente distribuídos em grandes áreas, sugerindo sincronizações entre centros de dados apenas quando as atualizações agregadas ultrapassam um certo limiar, devido as maiores restri-

ções de largura de banda. Conjuntamente com as atualizações síncronas e assíncronas, se é considerada a destilação. Este método é usado para reduzir a frequência de comunicação por meio do uso das saídas de previsão de um modelo para treinar outro modelo. Em (ANIL et al., 2018) propuseram incorporar a destilação no treinamento distribuído de DNNs. Cada dispositivo treina em um subconjunto dos dados e também utiliza as saídas de previsão de outros dispositivos para aprimorar o treinamento. Esse método permite a troca de informações com menor frequência, o que ajuda a evitar a comunicação excessiva de gradientes e pode ser particularmente útil em cenários com restrições de rede.

Dispositivos com conectividade ruim e alta latência podem resultar em atraso no treinamento distribuído.Para mitigar tal problema, (CHEN et al., 2016) sugere melhorias na SGD síncrona, como dispositivos de backup que computam atualizações de gradiente de dispositivos atrasados, permitindo que o treinamento prossiga sem esperar por todos, reduzindo assim a latência total.

5.4 Tamanho das Atualizações

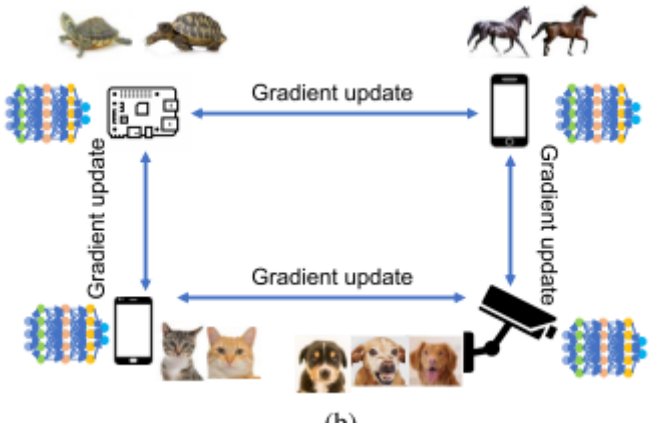
Assim como a frequência, o tamanho das atualizações de treinamento podem exigir muita largura de banda, o que é um desafio no cenário de edge computing, onde a largura de banda na última milha pode ser limitada. Para mitigar esse problema, técnicas de compressão de gradientes são usadas para reduzir o volume de dados enviados para o servidor central. Duas abordagens principais são a quantização de gradientes, que aproxima os gradientes em ponto flutuante usando números de baixa largura de bits, e a esparsificação de gradientes, que elimina gradientes menos importantes e comunica apenas os mais relevantes (LIN et al., 2017). Ambas as técnicas podem ser combinadas para melhorar a eficiência do treinamento, conforme mostrado em (LIN et al., 2017) aonde combinam a esparsificação de gradientes com outras técnicas, como correção de momento (QIAN, 1999) e aquecimento de treinamento (GOYAL et al., 2017), para acelerar a convergência do treinamento.

5.5 Protocolos de Comunicação Descentralizados

Até o momento, só foram discutidas arquiteturas de treinamento centralizadas, onde múltiplos dispositivos se conectam a um servidor de borda, garantindo que todos

os dispositivos convirjam para os mesmos parâmetros de modelo. No entanto, essa abordagem é limitada pela largura de banda do nó central. Para superar essa limitação, foi proposto um algoritmo de "fofoqueiro"descentralizado no qual a "fofoca"pode ser uma versão descentralizada da média elástica. Neste método, cada dispositivo computa suas próprias atualizações de gradiente e as comunica a alguns outros dispositivos, permitindo que eles cheguem a um consenso sobre o modelo de DNN. Como mostrado na Figura 7.3, a seguir. Em (BLOT et al., 2016) é proposto um algoritmo assíncrono de treinamento baseado em fofoca, que mostrou convergência mais rápida que a média elástica.

Figura 5.2: Treinamento Descentralizado



Fonte: (CHEN; RAN, 2019)

Em (JIN et al., 2016) investigaram a SGD baseada em fofoca para descobrir o quão escalável poderia ser, encontrando que métodos assíncronos convergem mais rapidamente com um número pequeno de trabalhadores, enquanto a SGD síncrona escalava melhor com um número maior de trabalhadores. No trabalho realizado em (LI et al., 2018), desenvolveram o INCEPTIONN, que combina compressão de gradientes e fofoca, dividindo os dispositivos em grupos onde cada dispositivo compartilha gradientes com o próximo no seu grupo, garantindo que todas as partes da DNN sejam atualizadas após várias iterações.

5.6 Treinamento Privado

Nesta seção, exploraremos os algoritmos de SGD, considerando as implicações de privacidade na comunicação de gradientes. Algo muito importante a ser comentado mas não foi o foco das experimentações. Mesmo que o treinamento local melhore a privacidade ao evitar o compartilhamento direto dos dados, os gradientes comunicados

ainda podem revelar dados privados (SHOKRI; SHMATIKOV, 2015), necessitando de técnicas adicionais para aumentar a privacidade.

Alguns estudos focaram no modelo de ameaça passivo, como dispositivos finais que seguem o protocolo de treinamento e tentam aprender sobre o modelo ou os dados observando as comunicações. Em (SHOKRI; SHMATIKOV, 2015) foi proposto a seleção de gradientes acima de um limiar e adicionar ruído antes de enviá-los, reduzindo vazamentos de informação. Já em (ABADI et al., 2016b) foi utilizado de recorte, média e adição de ruído para limitar a perda de privacidade no modelo geral.

Além de modificações no gradiente, a adição de ruído nos dados também já foi considerada. Conforme realizado em (ZHANG; HE; LEE, 2018) onde investigaram a adição de diferentes tipos de ruído aos dados de entrada, protegendo contra a descoberta de propriedades estatísticas de amostras individuais ou agregadas. Isso serve como forma de pré-processamento dos dados de treinamento, que pode fornecer proteção mesmo se o adversário obtiver acesso ao servidor de parâmetros e aos dados pós-processados.

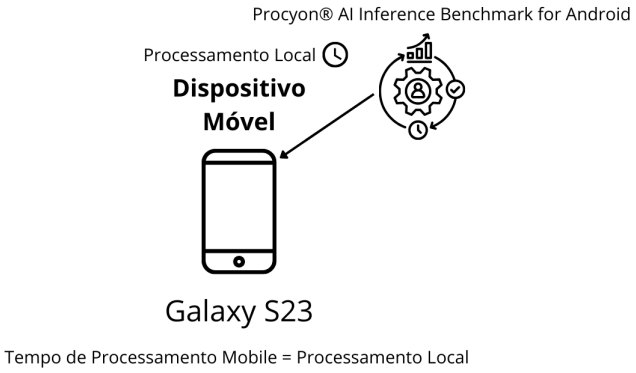
Um ótimo exemplo de computação segura foi realizado em (MOHASSEL; ZHANG, 2017) no qual se propôs um modelo de dois servidores que treinam uma rede neural com dados combinados de dispositivos finais, sem que os servidores aprendam nada além dos parâmetros da DNN. Um esquema baseado em regressão linear e logística segura de duas partes. O esquema inclui funções modificadas para *softmax* e *RELU* para eficiência, focando no treinamento de DNN ao invés de inferência.

6 METODOLOGIA

Para a realização deste trabalho, foram utilizados dois softwares de benchmark fornecidos pela UL Solutions, conforme detalhado na Seção 6.1. Esses softwares permitiram a coleta rápida e precisa dos dados de execução das redes neurais nos três dispositivos analisados. Os dados de latência de rede foram obtidos a partir do *dataset* apresentado na pesquisa de (CHARYYEV; ARSLAN; GUNES, 2020), conforme descrito na seção específica de Dados de Rede.

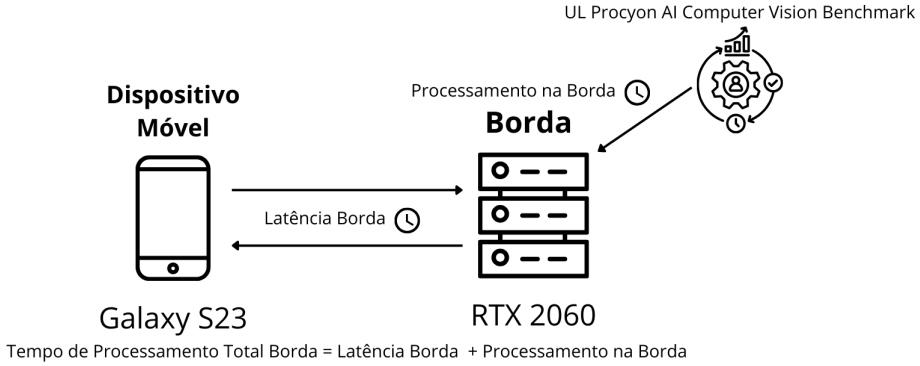
Com os tempos de execução em cada local e os dados de latência de rede, foi possível calcular o tempo total para cada cenário. Uma visão geral dos experimentos conduzidos é apresentada nos esquemas representativos: na Figura 6.1, o fluxo dos testes no dispositivo móvel; na Figura 6.2, o fluxo dos testes na borda; e na Figura 6.3, o fluxo dos testes na nuvem.

Figura 6.1: Experimento Local



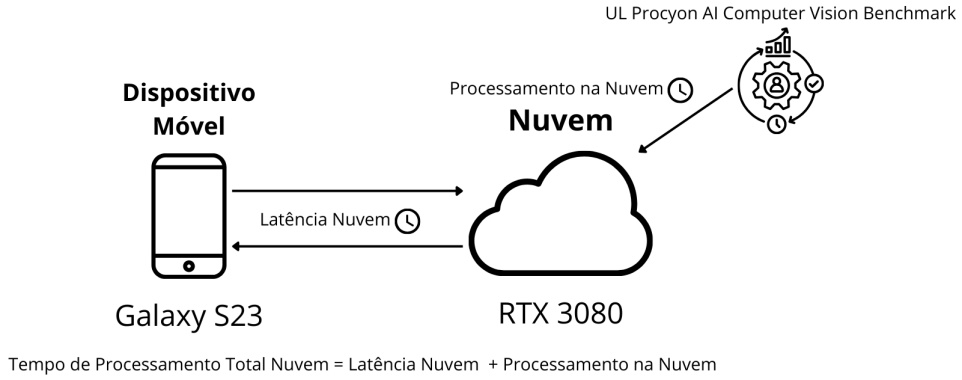
Fonte: Os Autores

Figura 6.2: Experimento Borda



Fonte: Os Autores

Figura 6.3: Experimento Nuvem



Fonte: Os Autores

6.1 UL Solutions

A escolha conjunta do *UL Procyon AI Computer Vision Benchmark* e do *Procyon® AI Inference Benchmark for Android* permitiu uma avaliação completa do desempenho de inferência de IA em diferentes sistemas operacionais e arquiteturas de maneira prática e eficiente, focando nos resultados e não na implementação das redes neurais. Esses dois benchmarks fazem parte do mesmo pacote fornecido pela UL Solutions, permitindo uma comparação entre diferentes plataformas. O *UL Procyon AI Computer Vision Benchmark* é projetado para testar dispositivos com sistemas operacionais Windows, oferecendo métricas detalhadas para tarefas de visão computacional em hardwares variados, como CPUs, GPUs e aceleradores de IA. Por outro lado, o *Procyon® AI Inference Benchmark for Android* é especificamente desenvolvido para medir o desempenho de inferência em dispositivos Android, onde a arquitetura de hardware e as otimizações de inferência são significativamente diferentes em relação aos sistemas Windows. O uso dessas duas ferramentas, que se complementam, viabilizou uma comparação abrangente e precisa, oferecendo percepções sobre o desempenho da IA em diferentes plataformas e facilitando a escolha da melhor configuração de hardware e software para atender às necessidades específicas de cada ambiente.

6.1.1 UL Procyon AI Computer Vision Benchmark

O UL Procyon AI Computer Vision Benchmark fornece informações sobre o desempenho de mecanismos de inferência de IA em ambiente Windows, auxiliando na es-

colha de quais motores suportar para otimizar o desempenho. Este benchmark apresenta uma variedade de mecanismos de inferência de IA de diferentes fornecedores, cujas pontuações refletem a eficiência das operações de inferência nos dispositivos. O Procyon AI Computer Vision Benchmark teve seu desenvolvimento e projeto realizado com parceiros da indústria por meio do UL Benchmark Development Program (BDP). As cargas de trabalho de IA envolvem tarefas comuns de visão computacional, como classificação de imagens, segmentação, detecção de objetos e super-resolução, sendo realizadas por uma série de redes neurais de alto desempenho, que podem ser processadas pela CPU, GPU ou um acelerador de IA dedicado. Os testes utilizam Kits de Desenvolvimento de Software (SDKs) como Microsoft® Windows ML, Qualcomm® SNPE, Intel® OpenVINO™, NVIDIA® TensorRT™ e Apple® Core ML™, oferecendo versões otimizadas em ponto flutuante e inteiro de cada modelo, o que possibilita uma análise detalhada das diferentes configurações de hardware. Nos nossos testes focamos na utilização do SDK NVIDIA® TensorRT™ em GPUs NVIDIA®.

O benchmark fornece uma pontuação final que é determinada a partir do tempo médio de inferência, aplicando uma fórmula que leva em consideração a média geométrica dos tempos médios de inferência de vários modelos mas utilzaremos apenas dos dados brutos sem considerar a pontuação. Os modelos de redes neurais utilizados e as suas respectivas entradas foram:

- MobileNet V3: Este modelo é projetado para dispositivos móveis, priorizando eficiência e desempenho. Seu tamanho de entrada é de 1x224x224x3, que representa uma imagem com largura e altura de 224 pixels e 3 canais de cor (RGB). A saída consiste em 1x1001, onde o modelo classifica a imagem em uma das 1001 categorias.
- Inception V4: Um modelo avançado para classificação de imagens, o Inception V4 é mais profundo e largo do que o MobileNet. A entrada tem dimensões de 1x299x299x3, permitindo um maior detalhamento na imagem, enquanto a saída também é 1x1001, semelhante ao MobileNet, fornecendo uma lista de probabilidades para a identificação de classes.
- YOLO V3: Este modelo é especializado em detecção de objetos e opera com uma entrada de 1x416x416x3. A saída do YOLO V3 é mais complexa, resultando em múltiplas dimensões (1x13x13x255, 1x26x26x255 e 1x52x52x255), onde cada conjunto de dimensões fornece informações sobre a localização e a confiança das detecções em diferentes escalas.

- DeepLab V3: Utilizado para segmentação de imagem, o DeepLab V3 recebe uma entrada de 1x513x513x3. A saída é uma máscara de segmentação com dimensões 1x513x513x21, onde cada pixel da imagem é classificado em uma das 21 categorias de objetos.
- ResNet50: Este modelo, baseado em uma arquitetura de rede residual, aceita uma entrada de 1x3x224x224, onde a primeira dimensão representa o número de imagens (1, neste caso), a segunda e a terceira representam a largura e a altura da imagem (224x224), e a quarta dimensão corresponde aos 3 canais de cor. A saída é 1x1000, representando as 1000 classes possíveis.
- Real-ESRGAN: Focado em super-resolução, este modelo tem como entrada uma imagem de 1x3x250x250 e produz uma saída com dimensões de 1x3x1000x1000, ampliando a imagem original de 250x250 pixels para 1000x1000 pixels, melhorando sua resolução e detalhes.representam cenários típicos de visão computacional, possibilitando comparações de desempenho relevantes.

6.1.2 Procyon® AI Inference Benchmark for Android

O *Procyon® AI Inference Benchmark* é uma ferramenta de avaliação desenvolvida para medir o desempenho e a precisão do hardware dedicado à inteligência artificial em dispositivos Android. O benchmark utiliza a Android Neural Networks API (NNAPI), detalhado na subseção 5.1.3, para acessar e avaliar a capacidade de processamento de IA nos dispositivos, o que permite aproveitar hardware especializado que realiza operações complexas de inferência de forma eficiente. São utilizadas uma gama de redes neurais de ponta, como MobileNet V3, Inception V4, SSDLite V3 e DeepLab V4. A instalação e o uso do benchmark são simples, permitindo que os usuários executem testes diretamente em seus dispositivos ou através da Android Debug Bridge (ADB), sem necessidade de configurações complicadas. Os resultados incluem métricas como tempos de inferência, consumo de energia e utilização de memória durante os testes, além de permitir a análise dos resultados em gráficos e tabelas. Essa funcionalidade é crucial para a otimização de drivers e para garantir que o hardware esteja sendo utilizado da melhor maneira possível em tarefas de inteligência artificial, ajudando as equipes de engenharia a identificar áreas de melhoria e maximizar a eficiência de seus dispositivos.

6.1.3 Android Neural Networks API

A NNAPI, ou Android Neural Networks API, é uma interface de programação projetada para permitir que aplicativos Android utilizem as capacidades de hardware de IA de forma eficiente e padronizada. Com o NNAPI, desenvolvedores podem acessar diretamente o potencial de processamento de redes neurais em dispositivos Android, utilizando diferentes tipos de hardware, como CPUs, GPUs, TPUs (Unidades de Processamento Tensorial) e NPUs (Unidades de Processamento Neural). Essa API possibilita que operações complexas, como convoluções e pooling, sejam realizadas em hardware especializado, aproveitando ao máximo o desempenho disponível em cada dispositivo. Além disso, a NNAPI facilita a criação de aplicativos portáveis que executam de forma consistente em dispositivos com diferentes configurações de hardware, abstraindo as diferenças e permitindo uma experiência mais uniforme ao usuário.

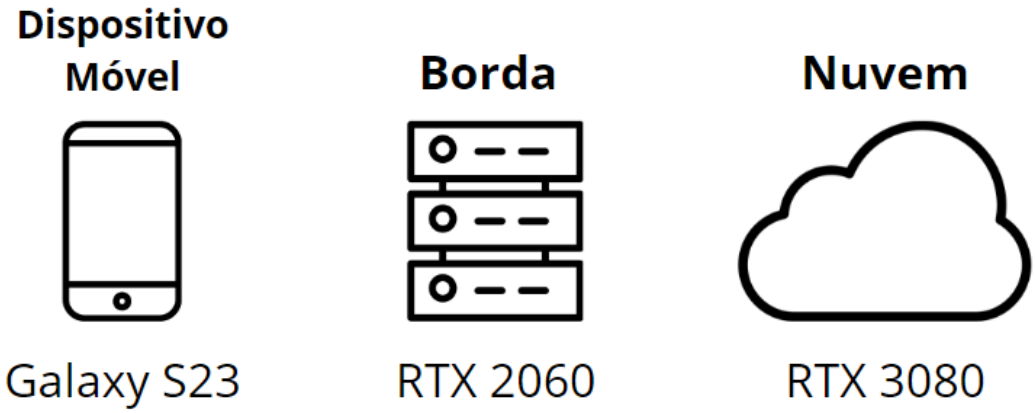
O *Procyon® AI Inference Benchmark for Android* utiliza a NNAPI para avaliar o desempenho de dispositivos Android ao executar inferência de redes neurais. Durante os testes, o benchmark faz uso da NNAPI para determinar automaticamente qual unidade de processamento deve ser utilizada para cada operação de IA, levando em consideração o tipo de tarefa, a eficiência energética e a capacidade do dispositivo. O *Procyon® AI Inference Benchmark for Android* coleta métricas detalhadas do desempenho de inferência ao delegar as operações para a GPU, CPU, ou, quando disponíveis, para NPUs e TPUs, priorizando o hardware mais adequado para maximizar o desempenho e a eficiência energética.A alocação de hardware também é influenciada pela carga de trabalho total do dispositivo, permitindo que o NNAPI distribua as tarefas para manter o equilíbrio e a desempenho do sistema, enquanto oferece aos desenvolvedores a flexibilidade de sugerir preferências de hardware conforme as necessidades específicas de suas aplicações.

6.2 Hardware Utilizado

Para a realização dos testes utilizamos como dispositivo mobile, Samsung Galaxy S23 Ultra, cujas especificações e detalhes estão na subseção 6.2.1. Sua escolha se deve ao fato de ser um dispositivo com alta capacidade, superando a média de dispositivos mobile utilizados pela populaçõa em geral, porém, sem o foco em IA conforme feito na linha Samsung Galaxy S24. Para simular a borda e a nuvem, foram utilizadas, respectivamente, uma GPU RTX 2060 e uma RTX 3080, detalhadas nas subseções 6.2.2 e 6.2.3.

Estas apresentam uma relação de ganho de desempenho ótimo, similar a borda e nuvem atualmente, e também foram de fácil acesso para a pesquisa.

Figura 6.4: Dispositivos Utilizados



Fonte: Os Autores

6.2.1 Dispositivo Móvel

O Samsung Galaxy S23 Ultra possui um conjunto robusto de hardware e software que permite a realização de processamento de inteligência artificial (IA) diretamente no dispositivo, conhecido como "on-device AI". Abaixo serão pontuados suas principais caracterísicas em relação ao trabalho realizado:

1. Processador (Chipset) com NPU Dedicada: O Snapdragon 8 Gen 2 for Galaxy é um chipset customizado pela Qualcomm para dispositivos Samsung Galaxy, oferecendo recursos avançados de inteligência artificial (IA) e aprendizado de máquina (ML) com uma Unidade de Processamento Neural (NPU) dedicada. Essa NPU é projetada para operações de IA em tempo real com eficiência energética, realizando tarefas complexas como reconhecimento de imagem, processamento de linguagem natural (para comandos de voz) e detecção de cenas fotográficas. Com capacidade de realizar trilhões de operações por segundo (TOPS). Além disso, o processador conta com uma CPU multicore composta por núcleos Kryo Prime, que alcançam até 3,36 GHz, núcleos Kryo Gold de 2,8 GHz (quad-core) e núcleos Kryo Silver de 2,0 GHz (tri-core). A CPU e a NPU trabalham em conjunto para dividir as operações: enquanto a NPU lida com tarefas intensivas de IA, a CPU gerencia operações secundárias, garantindo um sistema otimizado para atividades como otimização de

fotos e vídeos, realidade aumentada e gerenciamento inteligente de bateria.

2. GPU: A Adreno 740, é uma unidade gráfica poderosa e otimizada com suporte para inteligência artificial (IA) aplicada a processamento gráfico. Esse suporte permite que a Adreno 740 execute renderizações rápidas e eficientes, especialmente em jogos e aplicações de realidade aumentada(AR). A IA contribui para a otimização do desempenho gráfico e da eficiência de renderização, adaptando os recursos conforme as demandas visuais. Além disso, a GPU Adreno 740 oferece técnicas avançadas de upscaling e super-resolução, aprimorando a qualidade gráfica em tempo real. Essa funcionalidade é particularmente vantajosa em jogos com gráficos intensos, onde a IA aumenta a clareza e o detalhamento das imagens sem comprometer o desempenho.
3. Uso de IA: Com uma combinação poderosa de hardware, incluindo NPU, GPU e CPU, e software avançado, o Galaxy S23 Ultra executa o processamento de IA localmente, o que resulta em respostas rápidas, maior privacidade e menor consumo de dados. As capacidades de IA também suportam funcionalidades como fotografia aprimorada, assistentes de voz, segurança biométrica e otimização do sistema, tornando o Galaxy S23 Ultra uma plataforma poderosa para tecnologias baseadas em IA que vão além das necessidades convencionais de um smartphone.

6.2.2 Borda

Como dito no início desta seção, para simular a borda foi escolhida a GPU RTX 2060. Ela é baseada na arquitetura Turing que introduziu RT Cores para ray tracing em tempo real, apresenta também Tensor Cores utilizados para aceleração de operações em redes neurais. Os Tensor Cores, neste caso, conseguem realizar operações em precisão INT8, suportando FP16 com uma eficiência razoável e FP32, mas não tem núcleos otimizados como as GPUs da arquitetura Ampere. A taxa de FLOPS para FP32 é relativamente baixa em comparação com as GPUs mais recentes. Isso já a torna útil para tarefas de aprendizado de máquina, embora com limitações. Ou seja, RTX 2060 consegue lidar com tarefas de aprendizado de máquina em pequena escala, como treinamento e inferência em redes menores. Seus Tensor Cores permitem acelerar cálculos de redes neurais com precisão mista, mas a quantidade de memória e a largura de banda limitada fazem com que seu uso seja mais indicado para prototipagem ou projetos menos exigentes. A seguir na

Tabela 6.1, constam os dados da GPU.

Tabela 6.1: Arquitetura e Especificações de Hardware
RTX2060

Arquitetura	Turing (TU106)
Processo de Fabricação	12 nm
Núcleos CUDA	1920
Tensor Cores	240
RT Cores	30
Clock Base/Boost	1365 MHz / 1680 MHz
Memória	6 GB GDDR6, largura de banda de 336 GB/s
Consumo de Energia	160 W

Fonte: (Adrenaline, 2024)

6.2.3 Nuvem

A arquitetura Ampere, apresentada pela RTX 3080, traz uma evolução significativa em relação à Turing, com melhorias nos RT Cores e Tensor Cores, além de um aumento expressivo no número de núcleos CUDA. Essas mudanças permitem uma capacidade aprimorada de manipulação de dados para inteligência artificial (IA) e renderização de alta resolução, beneficiadas também pela inclusão da memória GDDR6X, que proporciona uma largura de banda maior. Com essas otimizações, o desempenho em precisão INT8 é elevado, facilitando a inferência de IA, uma área onde essa precisão é frequentemente utilizada. A RTX 3080 também se destaca no processamento FP16, com Tensor Cores de terceira geração projetados para operações de precisão mista (FP16/FP32), o que permite treinar redes neurais de forma mais eficiente. Em FP32, a arquitetura Ampere duplica a capacidade dos núcleos CUDA, que agora realizam duas operações FP32 por ciclo, aumentando o desempenho bruto em jogos e aplicações científicas. Essas inovações fazem da RTX 3080 uma escolha excelente para tarefas de aprendizado profundo, pois alia um maior throughput de FP32, eficiência em cálculos em FP16 e INT8, e uma capacidade robusta para lidar com redes neurais maiores e mais complexas. Segue na Tabela 6.2 os dados da GPU.

Tabela 6.2: Arquitetura e Especificações de Hardware
RTX3080

Arquitetura	Ampere(GA102)
Processo de Fabricação	8 nm
Núcleos CUDA	8704
Tensor Cores	272
RT Cores	68
Clock Base/Boost	1440 MHz / 1710 MHz
Memória	10 GB GDDR6X, largura de banda de 760 GB/s
Consumo de Energia	320 W

Para termos de comparação com uma GPU utilizada em um servidor de nuvem, segue uma breve descrição sobre a NVIDIA A100, e a Tabela 6.3 com seus dados lado a lado com as duas outras GPUs apresentadas.

Tabela 6.3: Comparativo - RTX 2060, RTX 3080 e NVIDIA A100

Especificação	**RTX 2060**	**RTX 3080**	**NVIDIA A100**
Arquitetura	Turing (TU106)	Ampere (GA102)	Ampere (GA100)
Processo de Fabricação	12 nm	8 nm	7 nm
Núcleos CUDA	1920	8704	8192 (80 GB)
Tensor Cores	240	272	432
RT Cores	30	68	Não aplicável
Clock Base/Boost	1365 MHz / 1680 MHz	1440 MHz / 1710 MHz	- / -
Memória	6 GB GDDR6	10 GB GDDR6X	80 GB HBM2e
Largura de Banda	336 GB/s	760 GB/s	2039 GB/s
Consumo de Energia	160 W	320 W	Até 400 W

A NVIDIA A100 foi projetada para atender às demandas de IA e aprendizado profundo em larga escala, com recursos excepcionais que a destacam para tarefas de inferência e treinamento de modelos complexos em data centers. Suas otimizações em operações INT4 e INT8 oferecem um desempenho acelerado em inferência, onde precisões mais baixas são suficientes para alcançar alta eficiência. Além disso, a A100 oferece desempenho incomparável em FP16, utilizando uma abordagem de precisão mista que combina FP16 com FP32, tornando-a altamente eficiente para treinamento e inferência de IA. A inovação do TensorFloat32 (TF32) permite realizar operações em alta precisão FP32 com o throughput de FP16, equilibrando velocidade e precisão durante o treinamento de redes neurais complexas. Para cálculos de alta precisão, como os necessários em simulações científicas e HPC, a A100 suporta FP64, o que a torna uma escolha ideal para ambientes

científicos e acadêmicos que exigem cálculos extremamente precisos. Equipadas com 40 GB ou 80 GB de memória HBM2e, as GPUs A100 suportam grandes modelos, como transformers e redes convolucionais em escala, sem a necessidade de dividir dados ou usar redes em paralelo, como ocorre nas GPUs de consumo. Assim, a A100 oferece um desempenho robusto e versátil em IA, aprendizado profundo e HPC, tornando-a essencial para projetos que exigem um equilíbrio entre precisão e velocidade.

7 RESULTADOS

Os resultados estão divididos em duas seções. Na primeira seção, são apresentados os tempos de processamento de cada dispositivo, acompanhados de um comparativo entre eles. Na segunda seção, são exibidos os valores de tempo de latência de rede durante a comunicação entre os dispositivos, bem como os tempos totais, resultantes da soma dos tempos de processamento dos dispositivos e dos tempos de latência de rede, considerando a modelagem aplicada.

7.1 Tempos de Inferência Locais

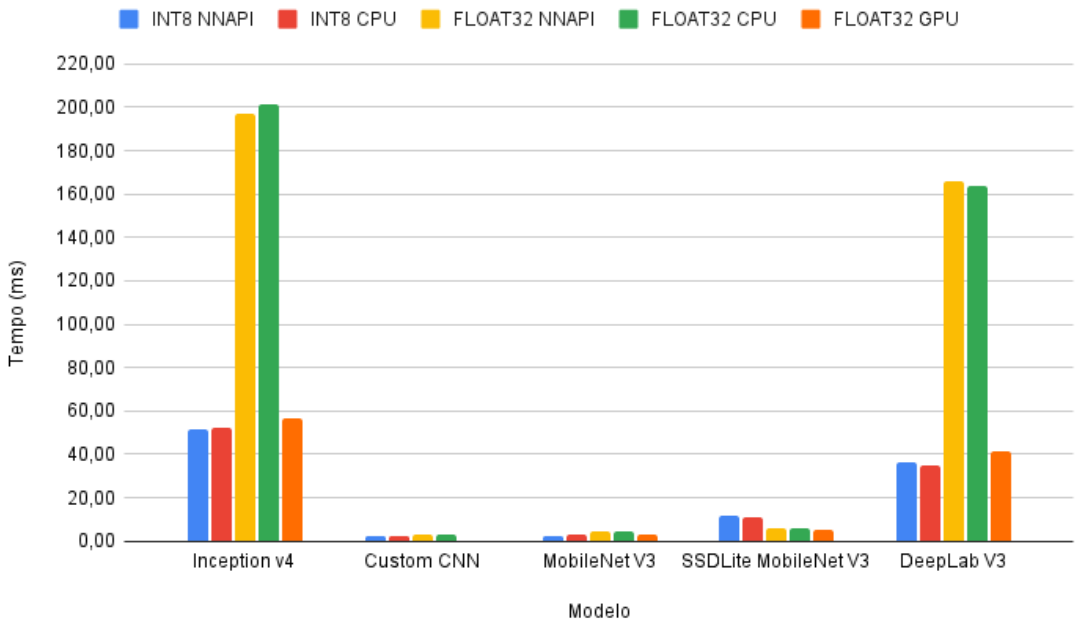
Os dados apresentados nesta seção foram obtidos a partir de testes realizados em três cenários distintos: dispositivo móvel (Samsung Galaxy S23), borda (RTX 2060) e nuvem (RTX 3080), utilizando benchmarks dedicados da UL Solutions para inferência de inteligência artificial. Esses resultados são provenientes de 10 execuções para os dispositivos de borda e nuvem com o UL Procyon AI Computer Vision Benchmark, e 5 execuções no dispositivo móvel utilizando o Procyon AI Inference Benchmark for Android. As métricas calculadas incluem tempo médio de inferência, o desvio padrão e a variação percentual, levando em consideração três precisões numéricas (INT8, Float16 e Float32). As tabelas com os valores obtidos de cada dispositivo estão ao final desta seção, indo da Tabela 7.1 até a Tabela 7.8.

7.1.1 Dispositivo Móvel

O gráfico apresentado na Figura 7.1 apresenta uma comparação do tempo de execução (em milissegundos) entre diferentes modelos de redes neurais no contexto mobile, considerando a interação entre precisão (INT8 e FLOAT32) e hardware utilizado (NNAPI, CPU, GPU). Observa-se que modelos mais complexos, como Inception v4 e DeepLab V3, exibem tempos significativamente mais altos ao usar FLOAT32, especialmente com NNAPI e CPU, refletindo a sobrecarga computacional causada por maior precisão. Por outro lado, a configuração INT8 no NNAPI apresenta tempos de inferência consistentes e baixos, com desvios padrão menores, como evidenciado no Inception v4, onde a inferência tem média de 51,86 ms e variação de apenas 2,03%, dados encontrados na Tabela

7.2. Modelos mais leves, como Custom CNN e MobileNet V3, mostram maior eficiência e menor variação, com destaque para tempos de inferência abaixo de 5 ms na maioria das configurações. Vale notar que o FLOAT32 na GPU, apesar de ser utilizado em dispositivos avançados como o Galaxy S23, exibe tempos elevados de inicialização, mostrado na Tabela 7.1, chegando a 621,80 ms em MobileNet V3, indicando que o uso de GPUs para FLOAT32 em redes neurais pode ser mais vantajoso em cenários de inferências mais longas e menos sensíveis à latência inicial.

Figura 7.1: Mobile: Hardware e Precisão X Rede Neural



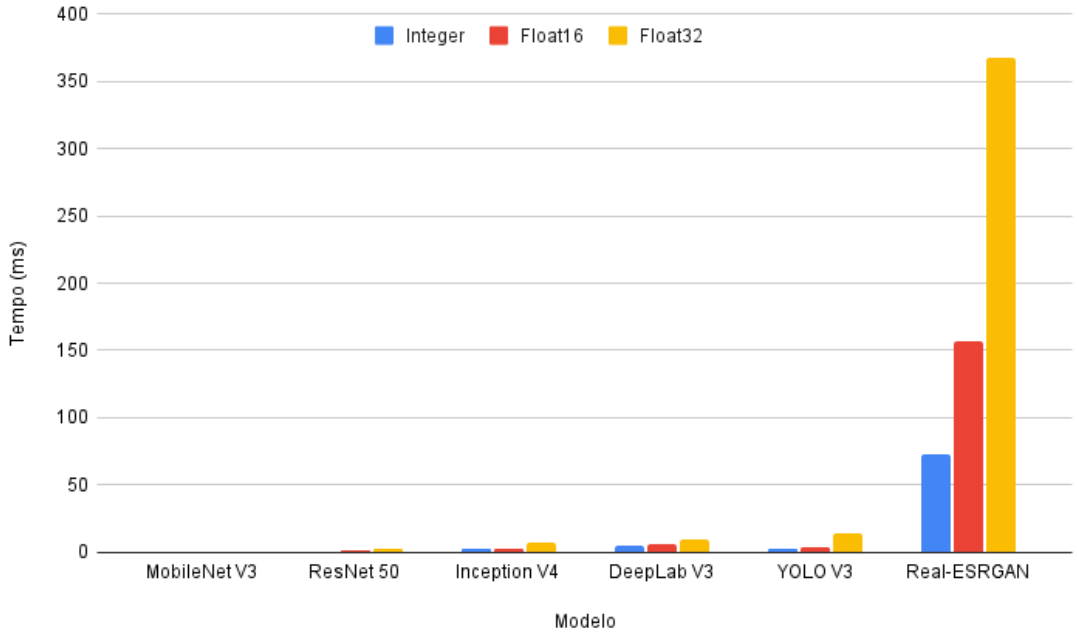
Fonte: Os Autores

7.1.2 Borda

Enquanto isso, o gráfico da Figura 7.2 mostra o tempo médio de inferência para diferentes modelos testados na GPU RTX 2060, considerando as precisões Integer, Float16 e Float32. Modelos leves como MobileNet V3 e ResNet 50 demonstram tempos de inferência extremamente baixos, com MobileNet V3 Integer apresentando uma média de apenas 0,6413 ms e desvio padrão de 2,57%, apresentado na Tabela 7.3, refletindo alta eficiência e consistência. Já modelos mais complexos, como Real-ESRGAN, exibem tempos significativamente mais elevados ao utilizar Float32 (367,91 ms), enquanto a precisão reduzida em Integer diminui drasticamente esse tempo para 72,35 ms.

Além disso, modelos intermediários, como YOLO V3, mostram variações maiores entre as precisões, com tempos de inferência aumentando de 3,21 ms (Integer) para 14,21 ms (Float32). No caso de modelos como Inception V4 e DeepLab V3, a transição para precisões mais altas também eleva substancialmente os tempos de inferência, indicando maior custo computacional associado à maior precisão. Essa análise reforça a importância de ajustar a precisão ao objetivo da aplicação, equilibrando desempenho e qualidade.

Figura 7.2: RTX 2060: Tempo Médio de Inferência x Rede Neural



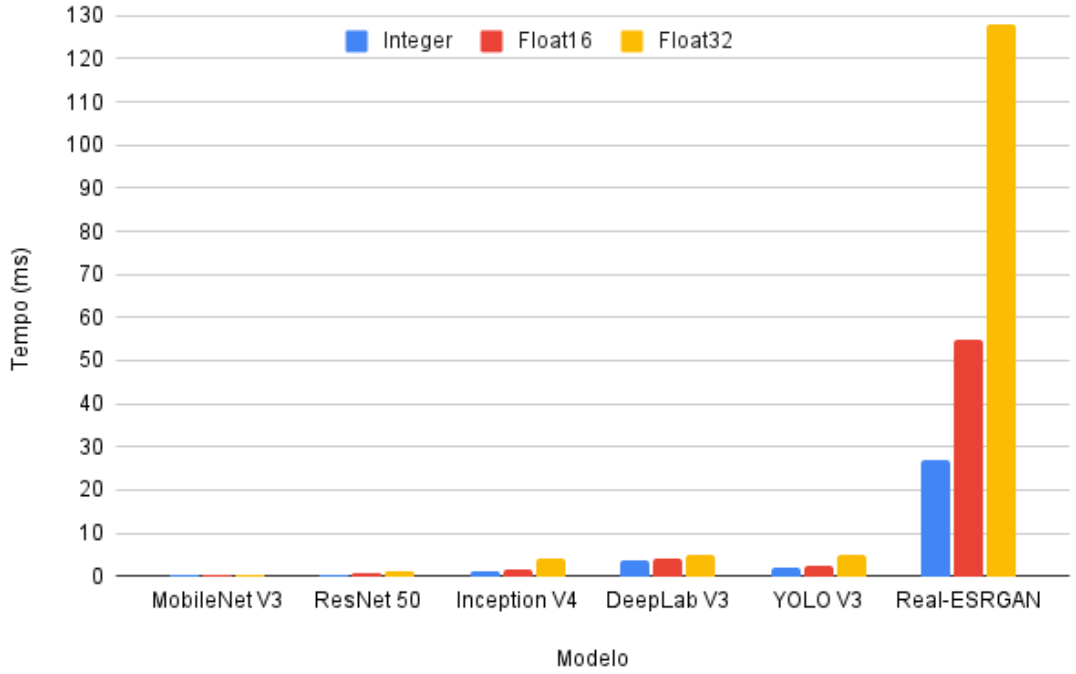
Fonte: Os Autores

7.1.3 Nuvem

No caso da nossa nuvem, com a GPU RTX 3080, analisando o gráfico da Figura 7.3, podemos perceber que modelos leves, como MobileNet V3 e ResNet 50, destacam-se pela alta eficiência, com tempos inferiores a 0,5 ms para as precisões Integer e Float16, evidenciando o desempenho da RTX 3080 em processamentos rápidos. A precisão Float32, embora mais lenta, mantém-se abaixo de 1 ms para esses modelos, mostrando boa consistência. Modelos mais complexos, como o Inception V4 e o DeepLab V3, apresentam uma diferença mais acentuada entre as precisões. O Inception V4 em Float32 requer cerca de 4 ms, enquanto Integer e Float16 ficam em torno de 1,3 ms e 1,5 ms, respectivamente. O DeepLab V3 também mostra resultados semelhantes, com Float32 alcançando 4,9 ms

contra valores mais baixos nas outras precisões. O destaque do gráfico, novamente, é o Real-ESRGAN, um modelo extremamente pesado, que registra 27 ms com precisão Integer, 55 ms com Float16 e 128 ms com Float32, refletindo o impacto da precisão nos tempos de execução. Apesar disso, o desvio padrão para todos os modelos é pequeno, indicando estabilidade nas execuções.

Figura 7.3: RTX 3080: Tempo Médio de Inferência x Rede Neural



Fonte: Os Autores

A seguir, estão as tabelas com os dados recolhidos e seus respectivos desvios padrão e porcentagem de variação.

Tabela 7.1: Tempo de Inicialização - Galaxy S23

Modelo	Configuração	Média Inicialização (ms)	Desvio Padrão Inicialização (ms)	Variação % Inicialização
Inception v4	INT8, NNAPI	12.4	4.84	39.04
	INT8, CPU	5.0	2.10	41.95
	FLOAT32, NNAPI	5.6	6.28	112.15
	FLOAT32, CPU	4.6	2.06	44.76
	FLOAT32, GPU	558.4	16.13	2.89
Custom CNN	INT8, NNAPI	0.2	0.40	200.00
	INT8, CPU	0.6	0.80	133.33
	FLOAT32, NNAPI	0.4	0.49	122.47
	FLOAT32, CPU	0.4	0.49	122.47
MobileNet V3	INT8, NNAPI	1.8	0.75	41.57
	INT8, CPU	1.2	0.75	62.36
	FLOAT32, NNAPI	1.8	1.72	95.58
	FLOAT32, CPU	1.6	0.49	30.62
	FLOAT32, GPU	621.8	112.27	18.06
SSDLite MobileNet V3	INT8, NNAPI	3.6	1.36	37.68
	INT8, CPU	3.8	0.75	19.69
	FLOAT32, NNAPI	4.6	2.42	52.53
	FLOAT32, CPU	2.4	1.02	42.49
	FLOAT32, GPU	675.2	4.53	0.67
DeepLab V3	INT8, NNAPI	4.0	3.03	75.83
	INT8, CPU	3.4	1.62	47.79
	FLOAT32, NNAPI	1.0	0.00	0.00
	FLOAT32, CPU	2.8	2.14	76.26
	FLOAT32, GPU	432.2	8.23	1.90

Tabela 7.2: Tempo de Inferência - Galaxy S23

Modelo	Configuração	Média Inferência (ms)	Desvio Padrão Inferência (ms)	Variação % Inferência
Inception v4	INT8, NNAPI	51.856	1.05	2.03
	INT8, CPU	52.298	1.28	2.44
	FLOAT32, NNAPI	197.246	2.92	1.48
	FLOAT32, CPU	201.300	1.98	0.99
	FLOAT32, GPU	56.948	0.16	0.28
Custom CNN	INT8, NNAPI	2.248	0.23	10.41
	INT8, CPU	2.256	0.24	10.83
	FLOAT32, NNAPI	3.216	0.07	2.11
	FLOAT32, CPU	3.262	0.09	2.76
MobileNet V3	INT8, NNAPI	2.688	0.09	3.20
	INT8, CPU	2.764	0.15	5.46
	FLOAT32, NNAPI	4.572	0.10	2.13
	FLOAT32, CPU	4.494	0.08	1.88
	FLOAT32, GPU	3.332	0.09	2.56
SSDLite MobileNet V3	INT8, NNAPI	11.424	0.15	1.32
	INT8, CPU	11.354	0.09	0.78
	FLOAT32, NNAPI	5.706	0.14	2.49
	FLOAT32, CPU	5.844	0.38	6.50
	FLOAT32, GPU	5.354	0.08	1.58
DeepLab V3	INT8, NNAPI	36.450	1.09	2.99
	INT8, CPU	35.168	0.27	0.77
	FLOAT32, NNAPI	165.608	6.35	3.83
	FLOAT32, CPU	163.976	2.94	1.79
	FLOAT32, GPU	41.332	0.44	1.06

Tabela 7.3: Tempo Médio de Inferência - RTX 2060

Modelo	**Precisão**	**Tempo Médio (ms)**	**Desvio Padrão (ms)**	**Variação (%)**
MobileNet V3	Integer	0,6413	0,0165	2,57
	Float16	0,6231	0,0062	0,99
	Float32	0,7539	0,01083	1,44
ResNet 50	Integer	0,8582	0,0201	2,34
	Float16	1,079	0,0035	0,32
	Float32	2,6053	0,01101	0,42
Inception V4	Integer	2,4819	0,0865	3,49
	Float16	2,6926	0,0127	0,47
	Float32	7,7616	0,03766	0,49
DeepLab V3	Integer	5,3204	0,4062	7,63
	Float16	6,773	0,0032	0,05
	Float32	9,7589	0,03236	0,33
YOLO V3	Integer	3,2122	0,0186	0,58
	Float16	4,425	0,0051	0,12
	Float32	14,2083	0,06252	0,44
Real-ESRGAN	Integer	72,3524	2,1966	3,04
	Float16	156,6296	0,1135	0,07
	Float32	367,9105	1,16172	0,32

Tabela 7.4: Tempo Mediano de Inferência - RTX 2060

Modelo	**Precisão**	**Tempo Mediano (ms)**	**Desvio Padrão (ms)**	**Variação (%)**
MobileNet V3	Integer	0,6187	0,0118	1,91
	Float16	0,5991	0,0042	0,7
	Float32	0,7388	0,00473	1
ResNet 50	Integer	0,8356	0,0107	1
	Float16	1,0681	0,0034	0,32
	Float32	2,5911	0,00635	0
Inception V4	Integer	2,4137	0,0146	1
	Float16	2,65	0,0053	0,2
	Float32	7,734	0,0126	0
DeepLab V3	Integer	5,269	0,4657	9
	Float16	6,754	0,0029	0,04
	Float32	9,7226	0,01756	0
YOLO V3	Integer	3,1565	0,1068	3
	Float16	4,4018	0,0054	0,12
	Float32	14,1717	0,0293	0
Real-ESRGAN	Integer	72,0687	2,2223	3
	Float16	156,061	0,0477	0,03
	Float32	366,7283	1,2765	0

Tabela 7.5: Contagem Total de Inferências - RTX 2060

Modelo	Precisão	Contagem Total de Inferências	Desvio Padrão	Variação (%)
MobileNet V3	Integer	65.889.700	1.434.150	2,18
	Float16	67.509.800	459.700	0,68
	Float32	59.203.000	744.967	1,26
ResNet 50	Integer	53.209.800	1.060.331	1,99
	Float16	44.799.000	147.650	0,33
	Float32	20.931.700	106.938	0,51
Inception V4	Integer	16.837.800	408.873	2,43
	Float16	15.861.700	121.090	0,76
	Float32	6.797.100	36.925	0,54
DeepLab V3	Integer	6.930.300	395.623	5,71
	Float16	5.919.000	10.500	0,18
	Float32	4.578.100	19.460	0,43
YOLO V3	Integer	11.348.400	35.722	0,31
	Float16	9.250.500	28.890	0,31
	Float32	3.692.800	17.423	0,47
Real-ESRGAN	Integer	825.800	23.907	2,9
	Float16	382.800	316	0,08
	Float32	163.300	640,31	0,39

Tabela 7.6: Tempo Médio de Inferência - RTX 3080

Modelo	**Precisão**	**Tempo Médio (ms)**	**Desvio Padrão (ms)**	**Variação (%)**
MobileNet V3	Integer	0,3698	0,00341	0,92
	Float16	0,3694	0,00789	2,14
	Float32	0,439	0,00446	1,02
ResNet 50	Integer	0,4628	0,00571	1,23
	Float16	0,5548	0,00418	0,75
	Float32	1,2284	0,00219	0,18
Inception V4	Integer	1,2816	0,01609	1,26
	Float16	1,4672	0,02152	1,47
	Float32	4,0188	0,00647	0,16
DeepLab V3	Integer	3,6526	0,01138	0,31
	Float16	4,0636	0,08183	2,01
	Float32	4,9164	0,01983	0,4
YOLO V3	Integer	1,8686	0,00768	0,41
	Float16	2,274	0,01847	0,81
	Float32	4,9522	0,00254	0,05
Real-ESRGAN	Integer	27,1374	0,06782	0,25
	Float16	54,8058	0,22395	0,41
	Float32	127,9466	0,17997	0,14

Tabela 7.7: Tempo Mediano de Inferência - RTX 3080

Modelo	Precisão	Tempo Mediano (ms)	Desvio Padrão (ms)	Variação (%)
MobileNet V3	Integer	0,3484	0,002	0,57
	Float16	0,352	0,00821	2,33
	Float32	0,4264	0,00561	1,32
ResNet 50	Integer	0,4452	0,00612	1,37
	Float16	0,5446	0,00325	0,6
	Float32	1,2256	0,00155	0,13
Inception V4	Integer	1,2392	0,0202	1,63
	Float16	1,43	0,00806	0,56
	Float32	4,0142	0,00712	0,18
DeepLab V3	Integer	3,5186	0,01642	0,47
	Float16	3,957	0,08675	2,19
	Float32	4,81	0,01954	0,41
YOLO V3	Integer	1,8242	0,00663	0,36
	Float16	2,2384	0,0147	0,66
	Float32	4,9308	0,00428	0,09
Real-ESRGAN	Integer	27,011	0,03554	0,13
	Float16	54,6144	0,18737	0,34
	Float32	127,8534	0,13392	0,1

Tabela 7.8: Contagem Total de Inferências - RTX 3080

Modelo	Precisão	Contagem Total de Inferências	Desvio Padrão	Variação (%)
MobileNet V3	Integer	95.410.400	179.864	0,19
	Float16	97.259.200	1.845.569	1,9
	Float32	87.007.200	1.023.054	1,18
ResNet 50	Integer	82.911.600	789.629	0,95
	Float16	74.628.400	820.189	1,1
	Float32	40.546.200	114.004	0,28
Inception V4	Integer	26.545.800	470.248	1,77
	Float16	24.277.600	174.314	0,72
	Float32	11.959.800	49.336	0,41
DeepLab V3	Integer	8.978.400	29.209	0,33
	Float16	8.448.800	88.938	1,05
	Float32	7.545.000	26.343	0,35
YOLO V3	Integer	15.802.000	174.320	1,1
	Float16	14.185.800	58.303	0,41
	Float32	8.678.800	9.570	0,11
Real-ESRGAN	Integer	2.171.800	12.182	0,56
	Float16	1.086.600	4.624	0,43
	Float32	468.000	748,33	0,16

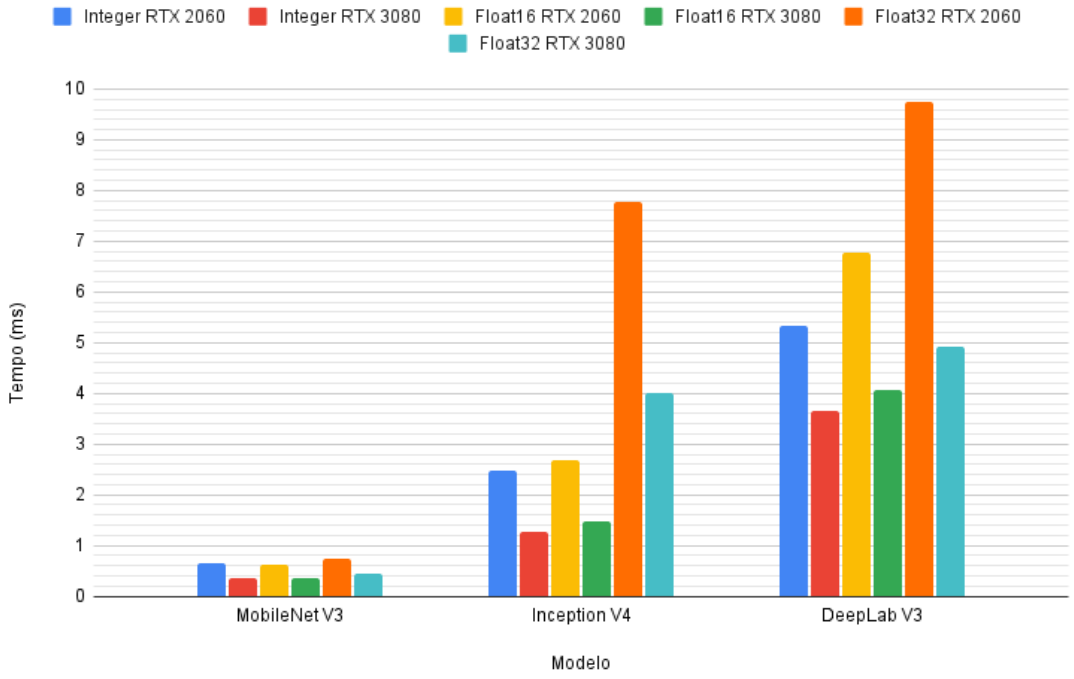
7.1.4 Borda x Nuvem

Comparando os resultados dos tempos de processamento, sem considerar a latência de rede e focando nos modelos neurais em comum entre os dispositivos, podemos observar, pelo gráfico da Figura 7.4, que, de forma geral, a **RTX 3080** apresenta uma vantagem significativa em desempenho, reduzindo os tempos médios de inferência em todos os modelos. Por exemplo, para o modelo **MobileNet V3**, o tempo médio de inferência com precisão **Integer** na RTX 2060 é de **0,6413 ms**, enquanto na RTX 3080 é de apenas **0,3698 ms**, representando uma redução de aproximadamente **42%**. Essa tendência se mantém em modelos mais complexos, como o **Real-ESRGAN**, onde a RTX 3080, utilizando **Float32**, atinge **127,95 ms** contra **367,91 ms** na RTX 2060, uma melhoria de cerca de **65%**.

Essa diferença de desempenho pode ser atribuída à arquitetura mais avançada da **RTX 3080**, que oferece maior número de núcleos CUDA e maior eficiência no processa-

mento de operações paralelas, especialmente.

Figura 7.4: Tempos de Inferência: RTX 2060 e RTX 3080



Fonte: Os Autores

7.1.5 Mobile x Borda x Nuvem

Realizando a mesma análise comparativa entre os três dispositivos e as precisões, é evidente a existência de diferenças significativas no desempenho de inferência, como pode ser visualizado nas Figuras 7.5 e 7.6. Para o modelo **MobileNet V3**, os tempos mais baixos ocorrem nas GPUs RTX, com a RTX 3080 (0,37s) sendo **42% mais rápida** que a RTX 2060 (0,64s) em **Integer**. Em dispositivos móveis, a NNAPI em **Integer** (2,69s) é apenas **2,5% mais rápida** que o CPU (2,76s), mas ambas são mais de **625% mais lentas** que a RTX 3080. Em **Float32**, a RTX 3080 (0,44s) é **41% mais rápida** que a RTX 2060 (0,75s), enquanto no mobile a GPU (3,33s) é **27% mais rápida** que a NNAPI (4,57s).

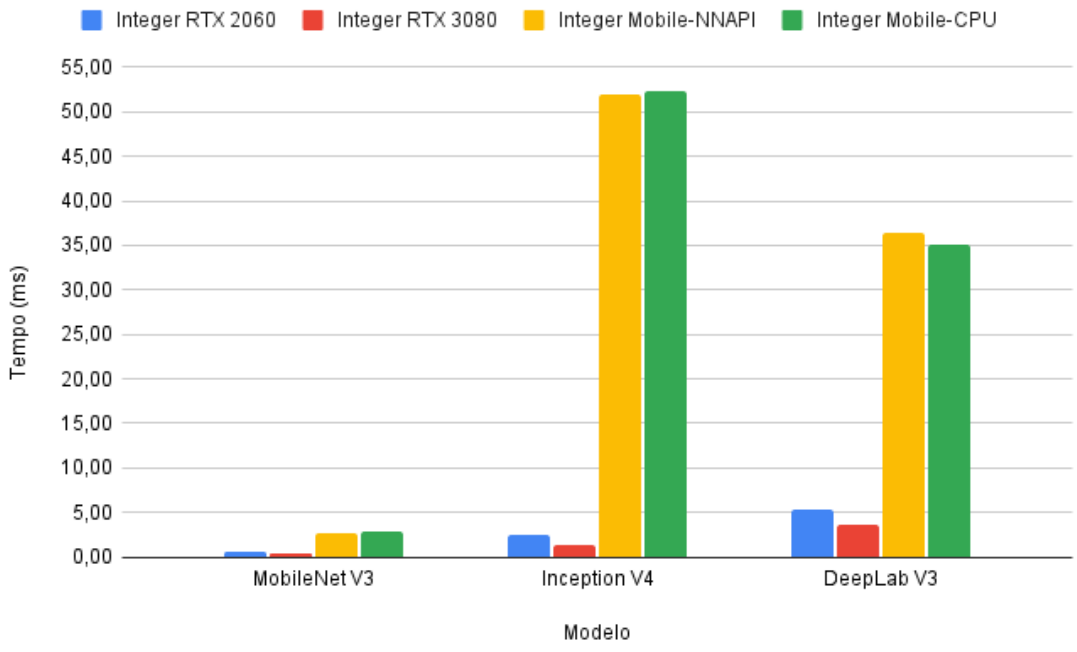
Para o modelo **Inception V4**, a RTX 3080 mantém a liderança em **Integer**, sendo **48% mais rápida** que a RTX 2060 (1,28s contra 2,48s). Em **Float32**, a RTX 3080 (4,02s) é novamente **48% mais rápida** que a RTX 2060 (7,76s). Em dispositivos móveis, os tempos em **Integer** (NNAPI: 51,86s, CPU: 52,30s) são praticamente equivalentes, com uma diferença de apenas **0,8%**, mas ambos são mais de **3.950% mais lentos** que a RTX

3080. Em **Float32**, a GPU móvel (56,95s) é **71% mais rápida** que a NNAPI (197,25s), mas ainda **1.317% mais lenta** que a RTX 3080.

No caso do modelo **DeepLab V3**, as GPUs RTX novamente se destacam, com a RTX 3080 (3,65s) sendo **31% mais rápida** que a RTX 2060 (5,32s) em **Integer**. Em **Float32**, a RTX 3080 (4,92s) é **50% mais eficiente** que a RTX 2060 (9,76s). Em dispositivos móveis, os tempos em **Integer** para NNAPI (36,45s) e CPU (35,17s) são comparáveis, com o CPU sendo apenas **3,5% mais rápido**, mas ambos são mais de **900% mais lentos** que a RTX 3080. Em **Float32**, a GPU móvel (41,33s) apresenta vantagem, sendo **75% mais rápida** que a NNAPI (165,61s), embora ainda seja **740% mais lenta** que a RTX 3080.

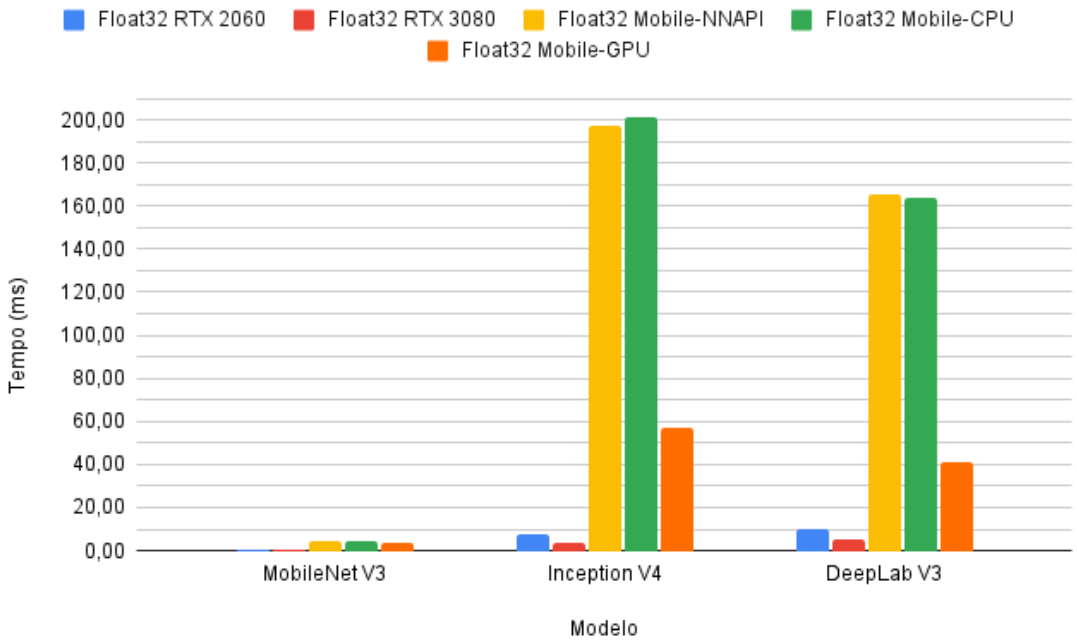
Esses resultados mostram que dispositivos de borda e nuvem oferecem desempenho significativamente superior, com ganhos expressivos de velocidade em relação aos dispositivos móveis, limitados principalmente em operações com maior precisão (**Float32**).

Figura 7.5: Comparativo Tempos de Inferência - Integer



Fonte: Os Autores

Figura 7.6: Comparativo Tempos de Inferência - Float 32



Fonte: Os Autores

Tabela 7.9: Tempos de Inferência por Dispositivo e Modelo (ms)

Dispositivo/Precisão	**MobileNet V3 (ms)**	**Inception V4 (ms)**	**DeepLab V3 (ms)**
Integer Mobile-NNAPI	2,69	51,86	36,45
Integer Mobile-CPU	2,76	52,30	35,17
Integer RTX 2060	0,64	2,48	5,32
Integer RTX 3080	0,37	1,28	3,65
Float32 Mobile-NNAPI	4,57	197,25	165,61
Float32 Mobile-CPU	4,49	201,30	163,98
Float32 Mobile-GPU	3,33	56,95	41,33
Float32 RTX 2060	0,75	7,76	9,76
Float32 RTX 3080	0,44	4,02	4,92

7.2 Latência de Rede

A partir deste momento, analisaremos o tempo total de processamento. Para isso, serão explicados os dados de latência de rede e como estes interferem no tempo total. Em seguida, serão analisados os tempos totais de execução, considerando a soma da latência

de rede com o tempo de processamento do dispositivo.

Os dados de latência de rede utilizados neste trabalho foram extraídos da pesquisa (CHARYYEV; ARSLAN; GUNES, 2020). Esta pesquisa envolveu medições em larga escala, realizadas com 8.456 nós RIPE Atlas (representando usuários finais), conectados a 6.341 servidores de borda da Akamai e 69 data centers de grandes provedores de nuvem, como Google Cloud, Amazon AWS, IBM Cloud, Oracle e Rackspace. Para cada nó, foram realizadas cinco medições de tempo de ida e volta (RTT) para cada destino, e a mediana dessas medições foi considerada como o valor representativo. Esse procedimento foi adotado tanto para as conexões à borda quanto à nuvem, garantindo consistência nos resultados.

Para os testes, os servidores de borda foram identificados por meio da resolução de nomes DNS de três grandes clientes da Akamai, sendo eles Apple, Microsoft e Yahoo. Isso permitiu localizar os endereços IP dos servidores mais próximos para cada usuário final. Por sua vez, a latência da nuvem foi medida implantando instâncias em cada data center dos provedores e selecionando a menor latência registrada entre os usuários e os servidores disponíveis.

Cada teste consistiu em cinco pings realizados entre os pontos de observação e os destinos (borda ou nuvem), utilizando a mediana dos valores como medida representativa da latência. A análise revelou que 58% dos usuários alcançaram servidores de borda com latência abaixo de 10 milissegundos, e 82% dentro de 20 milissegundos. Em contrapartida, apenas 29% dos usuários obtiveram uma latência menor que 10 milissegundos com servidores na nuvem, e 62% dentro de 20 milissegundos. Além disso, foi observado que, em 92% dos casos, a borda ofereceu latências mais baixas do que a nuvem. Contudo, em regiões como a Europa Ocidental, onde há maior densidade de data centers, a diferença de latência foi menos pronunciada. Por fim, foram identificados alguns casos atípicos em que a latência excedeu 250 milissegundos. Esses valores inesperados foram atribuídos a fatores como conexões VPN, uso de middleboxes ou configurações de rede incorretas.

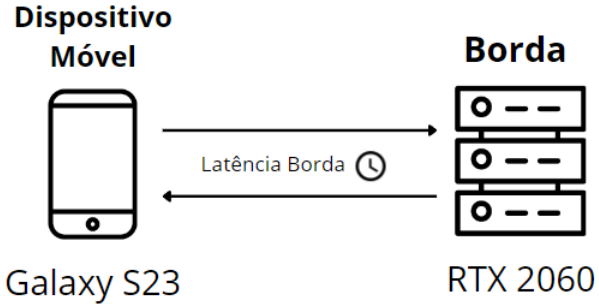
Vale ressaltar que, para os testes realizados, não foi considerado o tamanho das entradas na rede, sendo analisada apenas a latência do RTT.

7.2.1 Latência Borda

Os dados de latência da borda, Figura 7.7, foram coletados e processados por meio de um script que analisou arquivos CSV contendo os resultados dos testes de ping. Cada

arquivo foi lido para verificar se as colunas de resultados de latência estavam presentes. Em seguida, a mediana do RTT (tempo de ida e volta) foi calculada a partir dos cinco resultados de cada teste, associada ao IP alvo correspondente. Esse processo permitiu armazenar de forma padronizada os valores de latência de cada servidor de borda. Após a consolidação dos dados, o arquivo final foi analisado para calcular a latência média e a mediana de toda a amostra. A latência média encontrada foi de 22,324 ms, enquanto a mediana foi de 11,328 ms, refletindo a eficiência da borda em atender usuários com demandas de baixa latência.

Figura 7.7: Latência de Rede da Borda

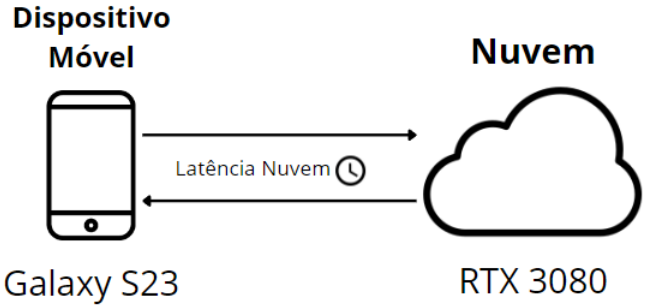


Fonte: Os Autores

7.2.2 Latência Nuvem

O processamento dos dados de latência da nuvem, Figura 7.8 seguiu um fluxo similar, mas com a adição de uma análise regional. Os arquivos CSV contendo os resultados das medições foram organizados de acordo com os continentes, permitindo associar as latências às respectivas localizações geográficas. Para cada arquivo, foi calculada a mediana da latência registrada, presentes na Tabela 7.10, e os resultados foram agrupados por continente, como pode ser visto na Tabela 7.11. A análise identificou variações regionais significativas, com latências menores em locais com maior densidade de data centers, como América do Norte e Europa Ocidental, e maiores em regiões como a Oceania, onde há pouca ou nenhuma presença de data centers.

Figura 7.8: Latência de Rede da Nuvem



Fonte: Os Autores

Tabela 7.10: Tempo Mediano de Latência por Nuvem

Nuvem	Tempo Mediano (ms)	Nuvem	Tempo Mediano (ms)
EuropeAmsterdam.csv	39,168	Queretaro-MEX01.csv	164,710
SaoPaola-SAO01.csv	217,695	EuropeSlough.csv	44,640
Sydney.csv	316,911	northamerica-northeast1.csv	114,343
Seoul-SEO01.csv	272,210	us-east1.csv	117,666
USWestNCalifornia.csv	172,681	Dallas-DAL01.csv	139,896
uk-london.csv	42,534	HongKong-HKG02.csv	253,593
europe-west1.csv	36,667	AustraliaSydney.csv	326,344
HongKong.csv	269,978	AsiaPacificSeoul.csv	293,107
europe-west3.csv	34,271	Chennai-CHE01.csv	189,079
Us-central1 2.csv	125,858	AsiaPacificSydney.csv	320,196
Amsterdam1-AMS01.csv	36,743	London-Lon02.csv	40,064
southamerica-east1.csv	233,047	USWestOregon.csv	183,190
USEastNViginia.csv	109,928	asia-east1.csv	273,371
Oslo-OSL01.csv	58,963	europe-north1.csv	53,766
AmericaSao$_{P}aulo.csv$	235,270	Us-east1 2.csv	108,194
EUFrankfurt.csv	34,034	Frankfurt5-FRA05.csv	35,008
Sydney-SYD01.csv	307,446	Milan-MIL01.csv	47,990
Montreal-MON01.csv	111,180	asia-southeast1.csv	307,724
europe-west4.csv	36,936	SanJose-SJC01.csv	166,956
Toronto-TOR01.csv	118,903	asia-south1.csv	368,234
Houston-HOU2.csv	142,514	Chicago.csv	123,472
Melbourne-MEL01.csv	310,472	asia-northeast1.csv	246,060
us-phoenix.csv	166,021	Seattle-SEA01.csv	161,014
EULondon.csv	40,010	Washington-WDC01.csv	106,884
Paris-PAR01.csv	40,659	CanadaCentral.csv	119,464
us-west1.csv	161,458	us-ashburn.csv	110,493
xLondon.csv	44,821	Dallas.csv	144,832
eu-frankfurt.csv	35,673	AsiaPacificSingapure.csv	267,801
AsiaPacificMumbai.csv	144,940	us-central1.csv	129,579
USEastOhio.csv	123,471	EUParis.csv	39,477
australia-southeast1.csv	296,202	EUIreland.csv	48,340
us-east4.csv	106,327	europe-west2.csv	40,885
NorthVirginia.csv	107,816	Singapore-SNG01.csv	226,206
SouthAmericaSaoPaola.csv	231,573	Tokyo-TOK02.csv	257,129
AsiaPacificTokyo.csv	264,827		

Tabela 7.11: Tempo Mediano de Latência por Continente

Continente	**Tempo Mediano (ms)**
Europa	40,010
América do Sul	231,573
América do Norte	123,471
Ásia	266,314
Oceania	308,959

Esses resultados destacam a eficácia da borda em oferecer baixa latência para a maioria dos usuários, mas também demonstram a necessidade de integração com a nuvem em cenários de alta demanda ou em áreas com menor cobertura de servidores.

7.2.3 Calculando o Tempo Total de Execução

Para determinar o tempo total necessário para realizar uma inferência, consideramos três cenários distintos. Primeiro, no **processamento local**, todo o trabalho é realizado no dispositivo móvel, resultando em um tempo total equivalente ao tempo de inferência no dispositivo:

$$T_{\text{local}} = T_{\text{mobile}}. \tag{7.1}$$

Segundo, no **processamento na borda**, o tempo total é composto pela soma da latência de rede entre o dispositivo móvel e a borda e o tempo de inferência realizado na borda:

$$T_{\text{borda}} = L_{\text{mobile-borda}} + T_{\text{borda}}. \tag{7.2}$$

Terceiro, no **processamento na nuvem**, o tempo total inclui a latência de rede entre o dispositivo móvel e a nuvem, que varia conforme o continente, e o tempo de inferência realizado na nuvem:

$$T_{\text{nuvem}} = L_{\text{mobile-nuvem}} + T_{\text{nuvem}}, \tag{7.3}$$

onde $L_{\text{mobile-nuvem}}$ corresponde à latência média medida para cada continente.

Com esses cálculos, é possível identificar o ponto em que cada abordagem se torna mais eficiente comparando os tempos totais. Para que o processamento na borda seja mais

vantajoso que o local, o tempo total na borda deve ser menor:

$$T_{\text{borda}} < T_{\text{local}} \quad \Rightarrow \quad L_{\text{mobile-borda}} + T_{\text{borda}} < T_{\text{mobile}}. \tag{7.4}$$

De forma semelhante, para que o processamento na nuvem seja mais vantajoso, o tempo total na nuvem deve ser inferior ao tempo local e ao tempo na borda. Assim, temos as condições:

$$T_{\text{nuvem}} < T_{\text{local}} \quad \Rightarrow \quad L_{\text{mobile-nuvem}} + T_{\text{nuvem}} < T_{\text{mobile}}, \tag{7.5}$$

$$T_{\text{nuvem}} < T_{\text{borda}} \quad \Rightarrow \quad L_{\text{mobile-nuvem}} + T_{\text{nuvem}} < L_{\text{mobile-borda}} + T_{\text{borda}}. \tag{7.6}$$

7.3 Tempo Total de Execução

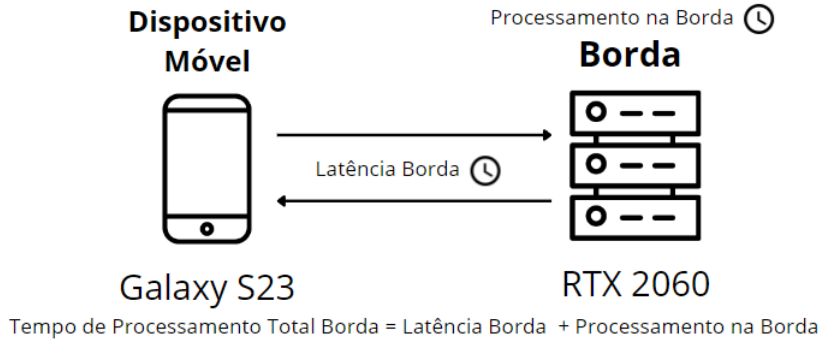
O principal objetivo deste trabalho é realizar uma análise detalhada do tempo total de execução de redes neurais em três dispositivos distintos: — mobile, edge e cloud — com foco na avaliação do processo de descarregamento (offloading) do processamento. Essa análise considera os modelos de redes compartilhados pelos dispositivos, bem como as diferentes precisões de cálculo utilizadas. Destaca-se que a precisão de ponto flutuante de 16 bits (float16), amplamente utilizada em dispositivos móveis, foi substituída pela precisão de 32 bits (float32) para garantir maior acurácia nos resultados. Neste contexto, dois cenários foram avaliados: o melhor caso, utilizando precisão inteira (int), que maximiza o desempenho computacional, e o pior caso, utilizando float32, permitindo observar como a maior precisão impacta o tempo de execução.

Conforme discutido na Subseção 7.2.3, para calcular o tempo total de execução, deve-se considerar a latência de rede entre o dispositivo móvel e a borda ou nuvem, dependendo do processo de offloading realizado. Por isso os dados a seguir consideram o tempo de execução do dispositivo mais o tempo de latência de rede conforme o local executado, cada cao exemplificado nas seguinte Figuras 7.9, 7.10 e 7.11.

Figura 7.9: Tempo Total de Processamento Local

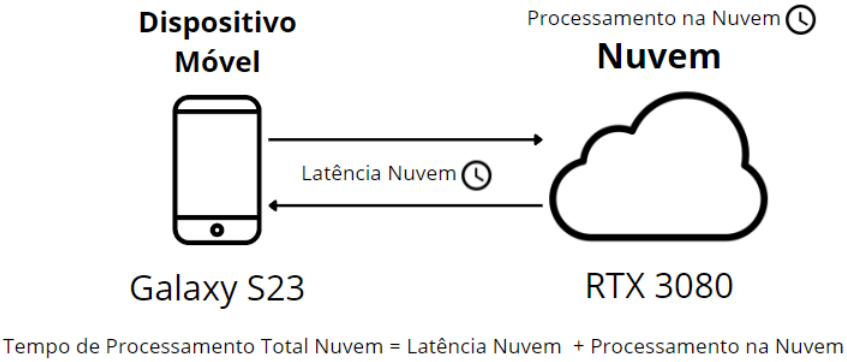
Fonte: Os Autores

Figura 7.10: Tempo Total de Processamento offloading para Borda



Fonte: Os Autores

Figura 7.11: Tempo Total de Processamento offloading para Nuvem



Fonte: Os Autores

Para redes leves, como o MobileNet V3, o dispositivo móvel (local) apresenta-se como a opção mais viável na maioria dos casos. Os valores observados estão apresentados nas Tabelas 7.12, 7.13 e 7.14.

Tabela 7.12: MobileNet V3 - Local (Dispositivo Móvel)

Precisão	**Método**	**Tempo (ms)**
Integer	NNAPI	2,69
Integer	CPU	2,76
Float32	NNAPI	4,57
Float32	CPU	4,49
Float32	GPU	3,33

Tabela 7.13: MobileNet V3 - Borda

Precisão	**Latência (ms)**	**Tempo Total (ms)**
Integer	Média	22,324 + 0,64 = 22,964
Integer	Mediana	11,328 + 0,64 = 11,968
Float32	Média	22,324 + 0,75 = 23,074
Float32	Mediana	11,328 + 0,75 = 12,078

Tabela 7.14: MobileNet V3 - Nuvem

Precisão	**Continente**	**Latência (ms)**	**Tempo Total (ms)**
Integer	Europa	40,01	40,010 + 0,37 = 40,380
Integer	América do Sul	231,573	231,573 + 0,37 = 231,943
Integer	América do Norte	123,47	123,471 + 0,37 = 123,841
Integer	Ásia	266,314	266,314 + 0,37 = 266,684
Integer	Oceania	308,959	308,959 + 0,37 = 309,329
Float32	Europa	40,01	40,010 + 0,44 = 40,450
Float32	América do Sul	231,573	231,573 + 0,44 = 232,013
Float32	América do Norte	123,471	123,471 + 0,44 = 123,911
Float32	Ásia	266,314	266,314 + 0,44 = 266,754
Float32	Oceania	308,959	308,959 + 0,44 = 309,399

O tempo total de processamento no dispositivo móvel com float32 varia entre $3,33$ ms (GPU) e $4,57$ ms (NNAPI), sendo substancialmente inferior ao tempo registrado na borda ($12,078$ ms) ou na nuvem, mesmo considerando a menor latência medida, como na Europa ($40,450$ ms). Em cenários reais, essa característica torna o processamento local ideal para aplicações sensíveis ao tempo, como sistemas embarcados, dispositivos IoT ou aplicações móveis que demandam baixa latência, tais como navegação em realidade aumentada.

Por outro lado, para redes mais complexas, como Inception V4 e DeepLab V3, observa-se um ponto de inflexão no qual a borda torna-se mais vantajosa. No caso do Inception V4, o tempo total na borda ($19,088$ ms com float32, conforme a Tabela 7.16) é significativamente menor do que no dispositivo móvel, onde o processamento local no GPU registra $56,95$ ms (Tabela 7.15). Apesar de apresentar o menor tempo de inferência, o móvel possui o maior tempo de inicialização (Tabela 7.1). Essa diferença evidencia que, para redes com tempos de inferência elevados, a borda oferece benefícios claros em aplicações que exigem respostas rápidas, como sistemas de vigilância ou análise de vídeo em tempo real. Similarmente, no caso do DeepLab V3, utilizado em tarefas como segmentação semântica, o cenário é análogo: enquanto o tempo total no móvel é $41,33$ ms (Tabela 7.18), na borda ele reduz para $21,088$ ms (Tabela 7.19), evidenciando sua adequação para cenários que demandam maior precisão computacional sem comprometer a latência de forma significativa.

Tabela 7.15: Inception V4 - Local (Dispositivo Móvel)

Precisão	**Método**	**Tempo (ms)**
Integer	NNAPI	51,86
Integer	CPU	52,3
Float32	NNAPI	197,25
Float32	CPU	201,3
Float32	GPU	56,95

Tabela 7.16: Inception V4 - Borda

Precisão	**Latência (ms)**	**Tempo Total (ms)**
Integer	Média	22,324 + 2,48 = 24,804
Integer	Mediana	11,328 + 2,48 = 13,808
Float32	Média	22,324 + 7,76 = 30,084
Float32	Mediana	11,328 + 7,76 = 19,088

Tabela 7.17: Inception V4 - Nuvem

Precisão	**Continente**	**Latência (ms)**	**Tempo Total (ms)**
Integer	Europa	40,01	40,010 + 1,28 = 41,290
Integer	América do Sul	231,573	231,573 + 1,28 = 232,853
Integer	América do Norte	123,471	123,471 + 1,28 = 124,751
Integer	Ásia	266,314	266,314 + 1,28 = 267,594
Integer	Oceania	308,959	308,959 + 1,28 = 310,239
Float32	Europa	40,01	40,010 + 4,02 = 44,030
Float32	América do Sul	231,573	231,573 + 4,02 = 235,593
Float32	América do Norte	123,471	123,471 + 4,02 = 127,491
Float32	Ásia	266,314	266,314 + 4,02 = 270,334
Float32	Oceania	308,959	308,959 + 4,02 = 312,979

Tabela 7.18: DeepLab V3 - Local (Dispositivo Móvel)

Precisão	Método	Tempo (ms)
Integer	NNAPI	36,45
Integer	CPU	35,17
Float32	NNAPI	165,61
Float32	CPU	163,98
Float32	GPU	41,33

Tabela 7.19: DeepLab V3 - Borda

Precisão	Latência (ms) Tempo Total (ms)		
Integer	Média	22,324	22,324 + 5,32 = 27,644
Integer	Mediana	11,328	11,328 + 5,32 = 16,648
Float32	Média	22,324	22,324 + 9,76 = 32,084
Float32	Mediana	11,328	11,328 + 9,76 = 21,088

Tabela 7.20: DeepLab V3 - Nuvem

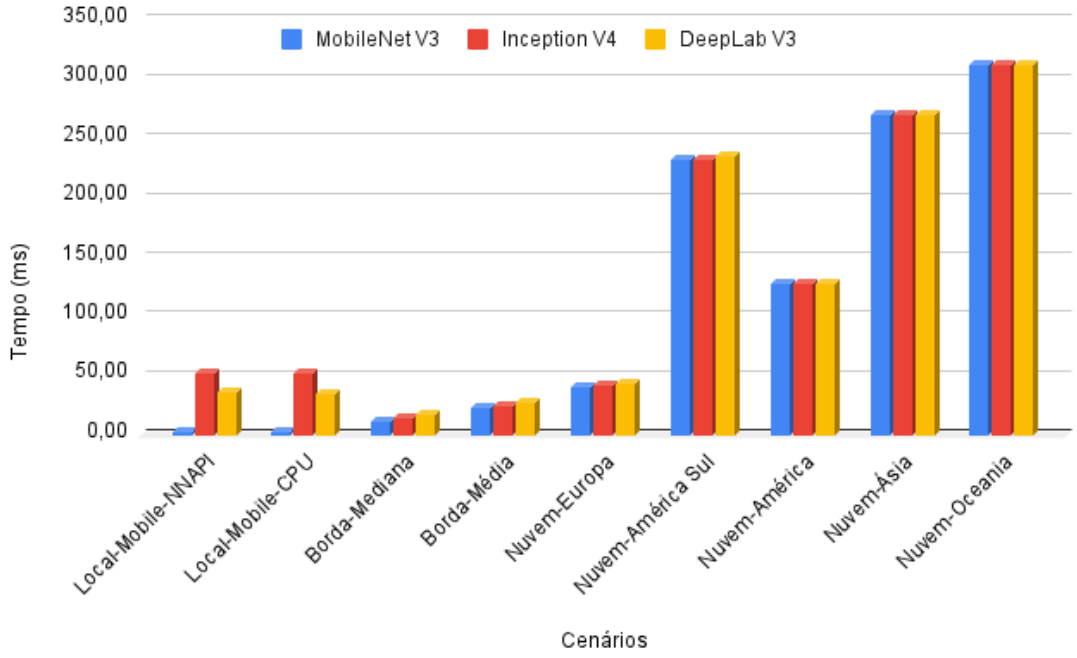
Precisão	Continente	Latência (ms)	Tempo Total (ms)
Integer	Europa	40,01	40,010 + 3,65 = 43,660
Integer	América do Sul	231,573	231,573 + 3,65 = 235,223
Integer	América do Norte	123,471	123,471 + 3,65 = 127,121
Integer	Ásia	266,314	266,314 + 3,65 = 269,964
Integer	Oceania	308,959	308,959 + 3,65 = 312,609
Float32	Europa	40,01	40,010 + 4,92 = 44,930
Float32	América do Sul	231,573	231,573 + 4,92 = 236,493
Float32	América do Norte	123,471	123,471 + 4,92 = 128,391
Float32	Ásia	266,314	266,314 + 4,92 = 271,234
Float32	Oceania	308,959	308,959 + 4,92 = 313,879

A nuvem, por sua vez, mostra-se viável apenas em situações nas quais a latência não seja um fator crítico. Mesmo utilizando a menor latência medida (Europa), os tempos totais para redes como Inception V4 e DeepLab V3 alcançam $44,030$ ms e $44,930$ ms, respectivamente, superando a borda em mais de duas vezes. Em regiões com latências maiores, como Oceania ou Ásia, esses tempos ultrapassam 300 ms, tornando a nuvem inadequada para qualquer aplicação em tempo real. Seu uso, portanto, é mais indicado para cenários que demandam alta capacidade de processamento, mas possuem pouca restrição temporal, como treinamento de redes neurais, análises preditivas em grandes volumes de dados ou processamento assíncrono.

Os pontos de inflexão, melhor visualizados nas Figuras 7.12 e 7.13, mostram que, para redes leves como o MobileNet V3, o dispositivo móvel é superior em qualquer cenário, desconsiderando o tempo de inicialização no caso da GPU, independentemente da latência. Para redes intermediárias, como Inception V4, a borda torna-se mais vantajosa quando a latência entre o dispositivo móvel e a borda permanece próxima à mediana ($11,328$ ms), mantendo o tempo total dentro de um intervalo aceitável. Por fim, a nuvem

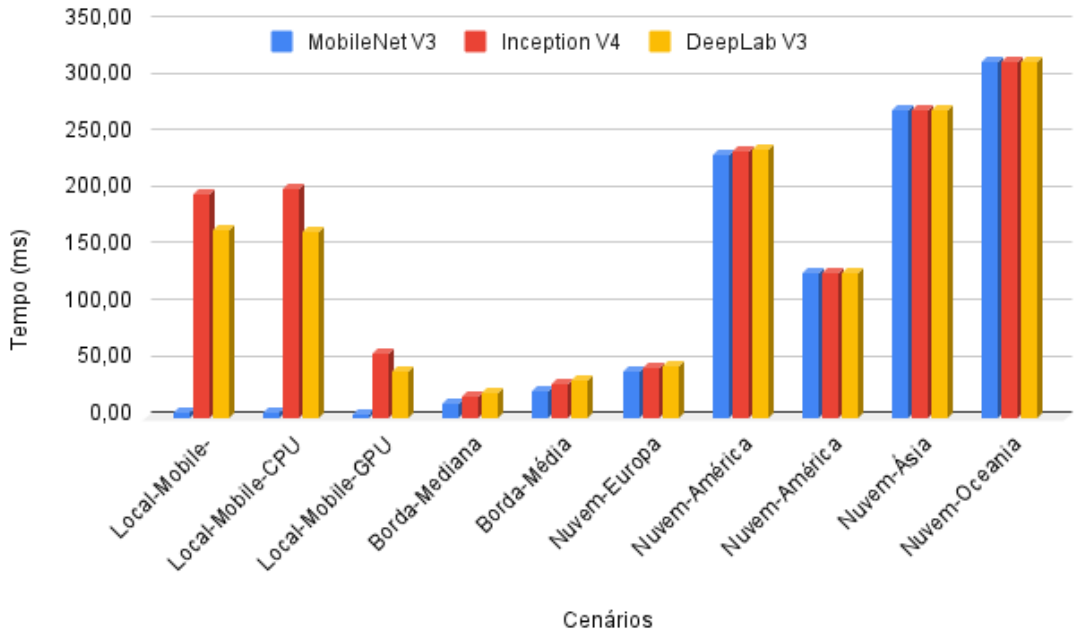
nunca supera os outros dois dispositivos em termos de tempo total, mesmo para redes pesadas, devido à sua dependência de latências elevadas.

Figura 7.12: Comparativo Tempos de Inferência Finais - Integer



Fonte: Os Autores

Figura 7.13: Comparativo Tempos de Inferência Finais - Float32



Fonte: Os Autores

8 CONCLUSÃO E CONSIDERAÇÕES FINAIS

O estudo explorou o desempenho das redes neurais no contínuo heterogêneo entre dispositivo móvel, borda e nuvem, destacando as diferenças dos tempos de execução entre essas arquiteturas.Ficou evidente que cada uma das plataformas apresenta vantagens e desvantagens específicas, que variam dependendo do cenário e dos requisitos de aplicação.

Sobre o desempenho nas inferências, é mostrado que os dispositivos móveis, embora limitados em recursos computacionais, podem ser adequados para inferência em redes neurais compactadas ou otimizadas, como MobileNet V3. No entanto, o tempo de execução e o consumo de energia foram significativamente maiores em redes mais complexas, como DeepLab V3 e Inception V4, quando comparados à borda e à nuvem.

Por outro lado, os servidores de borda se destacaram como uma solução intermediária, fornecendo latências mais baixas em comparação à nuvem, enquanto reduziam a sobrecarga no dispositivo móvel. Em particular, para aplicações sensíveis à latência, a borda demonstrou ser uma opção viável, especialmente em cenários onde a proximidade ao dispositivo final é crítica. Enquanto isso, a nuvem apresentou o maior poder computacional, permitindo a execução eficiente de redes complexas com menor tempo total de processamento. No entanto, os testes indicaram que a latência de rede ainda representa um desafio, especialmente para aplicações em tempo real que exigem respostas imediatas.

Em termos de implicações práticas, nossos resultados destacam a importância de selecionar a estratégia de execução mais apropriada, considerando as demandas específicas de cada aplicação. Para aplicações em tempo real os dispositivos móveis e servidores de tempo de resposta. Para redes complexas ou tarefas computacionalmente intensivas: A nuvem se torna indispensável, especialmente para cenários que exigem alta precisão e onde a latência não é um fator crítico. Pensando em privacidade e segurança, o uso da borda pode reduzir riscos associados à transmissão de dados para a nuvem, possibilitando maior controle sobre informações sensíveis.

Conclui-se que não existe uma solução única que se adeque a todos os cenários. Em vez disso, uma abordagem híbrida que aproveite as forças de dispositivos móveis, borda e nuvem pode maximizar o desempenho das redes neurais, reduzindo custos energéticos, trazendo mais segurança e melhorando a eficiência. Essa estratégia exige decisões dinâmicas, como o uso de algoritmos de particionamento e descarregamento inteligentes para otimizar a alocação de tarefas. Além disso, é reforçada a relevância do

desenvolvimento de tecnologias específicas para ambientes de borda, promovendo maior escalabilidade e eficiência.

REFERÊNCIAS

AAZAM, M.; HUH, E.-N. Fog computing micro datacenter based dynamic resource estimation and pricing model for iot. In: IEEE. **2015 IEEE 29th international conference on advanced information networking and applications**. [S.l.], 2015. p. 687–694.

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016.

ABADI, M. et al. Deep learning with differential privacy. In: **Proceedings of the 2016 ACM SIGSAC conference on computer and communications security**. [S.l.: s.n.], 2016. p. 308–318.

Adrenaline. **NVIDIA GeForce RTX 2060**. 2024. <https://www.adrenaline.com.br/produto/placa-de-video/nvidia-geforce-rtx-2060/>. Acessado em: 23 de outubro de 2024.

ANIL, R. et al. Large scale distributed neural network training through online distillation. **arXiv preprint arXiv:1804.03235**, 2018.

Baidu. **OpenEdge, Extend Cloud Computing, Data and Service Seamlessly to Edge Devices**. 2019. Online. Accessed: May. 30, 2024. Available from Internet: <https://github.com/baidu/openedge>.

BERNSTEIN, D. Containers and cloud: From lxc to docker to kubernetes. **IEEE cloud computing**, IEEE, v. 1, n. 3, p. 81–84, 2014.

BILAL, K. et al. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. **Computer Networks**, Elsevier, v. 130, p. 94–120, 2018.

BLOT, M. et al. Gossip training for deep learning. **arXiv preprint arXiv:1611.09726**, 2016.

BONOMI, F. et al. Fog computing and its role in the internet of things. In: **Proceedings of the first edition of the MCC workshop on Mobile cloud computing**. [S.l.: s.n.], 2012. p. 13–16.

CHARYYEV, B.; ARSLAN, E.; GUNES, M. H. Latency comparison of cloud datacenters and edge servers. In: **GLOBECOM 2020 - 2020 IEEE Global Communications Conference**. [S.l.]: IEEE, 2020. p. 1–6.

CHEN, J. et al. Revisiting distributed synchronous sgd. **arXiv preprint arXiv:1604.00981**, 2016.

CHEN, J.; RAN, X. Deep learning with edge computing: A review. **Proceedings of the IEEE**, v. 107, n. 8, p. 1655–1674, 2019.

CHEN, T. Y.-H. et al. Glimpse: Continuous, real-time object recognition on mobile devices. In: **Proceedings of the 13th ACM conference on embedded networked sensor systems**. [S.l.: s.n.], 2015. p. 155–168.

CHEN, Y. et al. Diannao family: energy-efficient hardware accelerators for machine learning. **Communications of the ACM**, ACM New York, NY, USA, v. 59, n. 11, p. 105–112, 2016.

COATES, A. et al. Deep learning with cots hpc systems. In: PMLR. **International conference on machine learning**. [S.l.], 2013. p. 1337–1345.

COLLOBERT, R.; BENGIO, S. Links between perceptrons, mlps and svms. In: **Proceedings of the twenty-first international conference on Machine learning**. [S.l.: s.n.], 2004. p. 23.

DAVIDSON, J. Cutting-edge ai innovations by intel, amd, and nvidia at ces 2024. **PC-Tablet**, 2024. Available from Internet: <https://www.pc-tablet.com/ces-2024-ai-innovations>.

DROLIA, U.; GUO, K.; NARASIMHAN, P. Precog: Prefetching for image recognition applications at the edge. In: **Proceedings of the Second ACM/IEEE Symposium on Edge Computing**. [S.l.: s.n.], 2017. p. 1–13.

DU, Z. et al. Shidiannao: Shifting vision processing closer to the sensor. In: **Proceedings of the 42nd annual international symposium on computer architecture**. [S.l.: s.n.], 2015. p. 92–104.

EdgeCortix. Edgecortix predicts next-gen ai will start to revolutionize business in 2024. **EdgeCortix**, 2024. Available from Internet: <https://www.edgecortix.com/news/2024-predictions>.

FACEBOOK. **Pytorch**. Accessed on May 20, 2024. Available from Internet: <https://pytorch.org>.

GOODFELLOW, I. et al. Generative adversarial nets. **Advances in neural information processing systems**, v. 27, 2014.

Google Cloud. **Edge TPU**. <https://cloud.google.com/edge-tpu/>. Accessed: 2024-06-15.

GOYAL, P. et al. Accurate, large minibatch sgd: Training imagenet in 1 hour. **arXiv preprint arXiv:1706.02677**, 2017.

HAN, S. et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In: **Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays**. [S.l.: s.n.], 2017. p. 75–84.

HAN, S.; MAO, H.; DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. **arXiv preprint arXiv:1510.00149**, 2015.

HAN, S. et al. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In: . New York, NY, USA: Association for Computing Machinery, 2016. ISBN 9781450342698. Available from Internet: <https://doi.org/10.1145/2906388.2906396>.

HINTON, G.; VINYALS, O.; DEAN, J. Distilling the knowledge in a neural network. **arXiv preprint arXiv:1503.02531**, 2015.

HINTON, G.; VINYALS, O.; DEAN, J. Distilling the knowledge in a neural network. **arXiv preprint arXiv:1503.02531**, 2015.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT press, v. 9, n. 8, p. 1735–1780, 1997.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.

HSIEH, K. et al. Gaia:{Geo-Distributed} machine learning approaching {LAN} speeds. In: **14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)**. [S.l.: s.n.], 2017. p. 629–647.

HUANG, J. et al. Speed/accuracy trade-offs for modern convolutional object detectors. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 7310–7311.

HUYNH, L. N.; LEE, Y.; BALAN, R. K. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In: **Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services**. [S.l.: s.n.], 2017. p. 82–95.

IANDOLA, F. N. et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. **arXiv preprint arXiv:1602.07360**, 2016.

JEANS, D. **Related's Hudson Yards: Smart City or Surveillance City?** 2019. Accessed on April, 2024. Available from Internet: <https://therealdeal.com/new-york/2019/03/15/hudson-yards-smart-city-or-surveillance-city/>.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. In: **Proceedings of the 22nd ACM international conference on Multimedia**. [S.l.: s.n.], 2014. p. 675–678.

JIANG, A. H. et al. Mainstream: Dynamic {Stem-Sharing} for {Multi-Tenant} video processing. In: **2018 USENIX Annual Technical Conference (USENIX ATC 18)**. [S.l.: s.n.], 2018. p. 29–42.

JIANG, J. et al. Chameleon: scalable adaptation of video analytics. In: **Proceedings of the 2018 conference of the ACM special interest group on data communication**. [S.l.: s.n.], 2018. p. 253–266.

JIANG, S. et al. Accelerating mobile applications at the network edge with software-programmable fpgas. In: IEEE. **IEEE INFOCOM 2018-IEEE Conference on Computer Communications**. [S.l.], 2018. p. 55–62.

JIN, P. H. et al. How to scale distributed deep learning? **arXiv preprint arXiv:1611.04581**, 2016.

KANG, Y. et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. Association for Computing Machinery, New York, NY, USA, v. 45, n. 1, 2017. ISSN 0163-5964. Available from Internet: <https://doi.org/10.1145/3093337.3037698>.

KHELIFI, H. et al. Bringing deep learning at the edge of information-centric internet of things. **IEEE Communications Letters**, v. 23, n. 1, p. 52–55, 2019.

LAI, L.; SUDA, N. Enabling deep learning at the lot edge. In: IEEE. **2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.], 2018. p. 1–6.

LANE, N. D. et al. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In: IEEE. **2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)**. [S.l.], 2016. p. 1–12.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group UK London, v. 521, n. 7553, p. 436–444, 2015.

LI, D. et al. Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition. In: IEEE. **2016 IEEE/ACM Symposium on Edge Computing (SEC)**. [S.l.], 2016. p. 64–76.

LI, G. et al. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In: SPRINGER. **Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27**. [S.l.], 2018. p. 402–411.

LI, H.; OTA, K.; DONG, M. Learning iot in edge: Deep learning for the internet of things with edge computing. **IEEE network**, IEEE, v. 32, n. 1, p. 96–101, 2018.

LI, Y. et al. A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks. In: IEEE. **2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)**. [S.l.], 2018. p. 175–188.

LIN, Y. et al. Deep gradient compression: Reducing the communication bandwidth for distributed training. **arXiv preprint arXiv:1712.01887**, 2017.

LIU, C. et al. A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. **IEEE Transactions on Services Computing**, IEEE, v. 11, n. 2, p. 249–261, 2017.

LIU, S. et al. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In: **Proceedings of the 16th annual international conference on mobile systems, applications, and services**. [S.l.: s.n.], 2018. p. 389–400.

LIU, W. et al. Ssd: Single shot multibox detector. In: SPRINGER. **Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14**. [S.l.], 2016. p. 21–37.

MANNING, C.; SCHUTZE, H. **Foundations of statistical natural language processing**. [S.l.]: MIT press, 1999.

MAO, J. et al. Modnn: Local distributed mobile computing system for deep neural network. In: IEEE. **Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017**. [S.l.], 2017. p. 1396–1401.

Microsoft. **Azure IoT Edge, Extend Cloud Intelligence and Analytics to Edge Devices**. 2019. Online. Accessed: May. 30, 2024. Available from Internet: <https://github.com/Azure/iotedge>.

MOHASSEL, P.; ZHANG, Y. Secureml: A system for scalable privacy-preserving machine learning. In: IEEE. **2017 IEEE symposium on security and privacy (SP)**. [S.l.], 2017. p. 19–38.

MORITZ, P. et al. Sparknet: Training deep networks in spark. **arXiv preprint arXiv:1511.06051**, 2015.

MOURADIAN, C. et al. A comprehensive survey on fog computing: State-of-the-art and research challenges. **IEEE communications surveys & tutorials**, IEEE, v. 20, n. 1, p. 416–464, 2017.

MUDASSAR, B. A.; KO, J. H.; MUKHOPADHYAY, S. Edge-cloud collaborative processing for intelligent internet of things: A case study on smart surveillance. In: **Proceedings of the 55th Annual Design Automation Conference**. New York, NY, USA: Association for Computing Machinery, 2018. (DAC '18). ISBN 9781450357005. Available from Internet: <https://doi.org/10.1145/3195970.3196036>.

NVIDIA. **Cuda**. Accessed on May 20, 2024. Available from Internet: <https://developer.nvidia.com/cuda-zone>.

NVIDIA. **Cudnn**. Accessed on May 20, 2024. Available from Internet: <https://developer.nvidia.com/cudnn>.

PAPINENI, K. et al. Bleu: a method for automatic evaluation of machine translation. In: **Proceedings of the 40th annual meeting of the Association for Computational Linguistics**. [S.l.: s.n.], 2002. p. 311–318.

PATEL, M. et al. Mobile-edge computing introductory technical white paper. **White paper, mobile-edge computing (MEC) industry initiative**, v. 29, p. 854–864, 2014.

QIAN, N. On the momentum term in gradient descent learning algorithms. **Neural networks**, Elsevier, v. 12, n. 1, p. 145–151, 1999.

Qualcomm. Qualcomm brings the best of on-device ai to more smartphones with snapdragon 8s gen 3. **Qualcomm**, 2024. Available from Internet: <https://www.qualcomm.com/news/releases/2024/03/qualcomm-brings-the-best-of-on-device-ai-to-more-smartphones-with-snapdragon-8s-gen-3>.

Qualcomm. Qualcomm continues to bring the generative ai revolution to devices and empowers developers with qualcomm ai hub. **Qualcomm**, 2024. Available from Internet: <https://www.qualcomm.com/news/releases/2024/02/qualcomm-continues-to-bring-the-generative-ai-revolution-to-devices>.

RAN, X. et al. Deepdecision: A mobile deep learning framework for edge video analytics. In: **IEEE INFOCOM 2018 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2018. p. 1421–1429.

REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 7263–7271.

RUDER, S. An overview of gradient descent optimization algorithms. **arXiv preprint arXiv:1609.04747**, 2016.

RUSSAKOVSKY O., D.-J. S. H. Imagenet large scale visual recognition challenge. **Int. J. Comput. Vis., vol. 115, no. 3,**, 2015.

SATYANARAYANAN, M. The emergence of edge computing. **Computer**, v. 50, n. 1, p. 30–39, 2017.

SATYANARAYANAN, M. et al. The case for vm-based cloudlets in mobile computing. **IEEE Pervasive Computing**, v. 8, n. 4, p. 14–23, 2009.

SHI, W. et al. Edge computing: Vision and challenges. **IEEE internet of things journal**, Ieee, v. 3, n. 5, p. 637–646, 2016.

SHOKRI, R.; SHMATIKOV, V. Privacy-preserving deep learning. In: **Proceedings of the 22nd ACM SIGSAC conference on computer and communications security**. [S.l.: s.n.], 2015. p. 1310–1321.

TAN, M. et al. Mnasnet: Platform-aware neural architecture search for mobile. In: **Proceedings of the IEEE/CVF conference on computer vision and pattern recognition**. [S.l.: s.n.], 2019. p. 2820–2828.

TEERAPITTAYANON, S.; MCDANEL, B.; KUNG, H.-T. Branchynet: Fast inference via early exiting from deep neural networks. In: IEEE. **2016 23rd international conference on pattern recognition (ICPR)**. [S.l.], 2016. p. 2464–2469.

TEERAPITTAYANON, S.; MCDANEL, B.; KUNG, H.-T. Distributed deep neural networks over the cloud, the edge and end devices. In: IEEE. **2017 IEEE 37th international conference on distributed computing systems (ICDCS)**. [S.l.], 2017. p. 328–339.

VASWANI, A. et al. Attention is all you need. In: GUYON, I. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2017. v. 30. Available from Internet: <https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.

WANG, S. et al. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In: IEEE. **IEEE INFOCOM 2018-IEEE conference on computer communications**. [S.l.], 2018. p. 63–71.

WANG, X. et al. Convergence of edge computing and deep learning: A comprehensive survey. **IEEE Communications Surveys Tutorials**, v. 22, n. 2, p. 869–904, 2020.

WANG, X.; WANG, X.; MAO, S. Rf sensing in the internet of things: A general deep learning framework. **IEEE Communications Magazine**, IEEE, v. 56, n. 9, p. 62–67, 2018.

XIONG, Y. et al. Extend cloud to edge with kubeedge. In: IEEE. **2018 IEEE/ACM Symposium On Edge Computing (SEC)**. [S.l.], 2018. p. 373–377.

YANG, Y. Multi-tier computing networks for intelligent iot. **Nature Electronics**, Nature Publishing Group UK London, v. 2, n. 1, p. 4–5, 2019.

YAO, S. et al. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In: **Proceedings of the 15th ACM conference on embedded network sensor systems**. [S.l.: s.n.], 2017. p. 1–14.

YOUSEFPOUR, A. et al. All one needs to know about fog computing and related edge computing paradigms: A complete survey. **Journal of Systems Architecture**, v. 98, p. 289–330, 2019. ISSN 1383-7621. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1383762118306349>.

YUGATECH. Qualcomm snapdragon vs mediatek dimensity in 2024: What's your pick? **YugaTech**, 2024. Available from Internet: <https://www.yugatech.com/qualcomm-snapdragon-vs-mediatek-dimensity-in-2024-whats-your-pick/>.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: SPRINGER. **Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13**. [S.l.], 2014. p. 818–833.

ZHANG, H. et al. Live video analytics at scale with approximation and {Delay-Tolerance}. In: **14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)**. [S.l.: s.n.], 2017. p. 377–392.

ZHANG, T.; HE, Z.; LEE, R. B. Privacy-preserving machine learning through data obfuscation. **arXiv preprint arXiv:1807.01860**, 2018.

ZHAO, Z.; BARIJOUGH, K. M.; GERSTLAUER, A. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 37, n. 11, p. 2348–2359, 2018.