**UNIVERSIDADE FEDERAL DO CEARÁ**

**CENTRO DE CIÊNCIAS**

**DEPARTAMENTO DE COMPUTAÇÃO**

**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO**

**CLARO HENRIQUE SILVA SALES**

# ANÁLISE DE DESEMPENHO DA QUANTIZAÇÃO EM SOLUÇÕES DE APRENDIZADO DIVIDIDO FEDERADO

**FORTALEZA**

**2025**

CLARO HENRIQUE SILVA SALES

# ANÁLISE DE DESEMPENHO DA QUANTIZAÇÃO EM SOLUÇÕES DE APRENDIZADO DIVIDIDO FEDERADO

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Computação de Alto Desempenho

Orientador: Prof. Dr. Francisco Heron de Carvalho Junior

FORTALEZA

2025

CLARO HENRIQUE SILVA SALES

ANÁLISE DE DESEMPENHO DA QUANTIZAÇÃO EM SOLUÇÕES DE APRENDIZADO DIVIDIDO FEDERADO

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Computação de Alto Desempenho

Aprovada em: 28 de Novembro de 2025.

BANCA EXAMINADORA

---

Prof. Dr. Francisco Heron de Carvalho Junior (Orientador)
Universidade Federal do Ceará (UFC)

---

Prof. Dr. Dalvan Jair Griebler
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

---

Prof. Dr. César Lincoln Cavalcante Mattos
Universidade Federal do Ceará (UFC)

---

Prof. Dr. Allberson Bruno de Oliveira Dantas
Universidade da Integração Internacional da Lusofonia Afro-Brasileira (UNILAB)

## AGRADECIMENTOS

Aos meus pais, Marineide Silva e Francisco Rômulo, por todo o apoio, afeto e pelos sacrifícios que fizeram ao longo de toda a minha vida.

Ao meu orientador, Prof. Dr. Francisco Heron de Carvalho Júnior, pela excelente orientação, paciência, broncas e discussões técnicas necessárias para conclusão desse trabalho.

Aos membros da banca examinadora, ao Prof. Dr. Allberson Bruno de Oliveira Dantas e Prof. Dr. César Lincoln Cavalcante Mattos por terem acompanhado e contribuído em meu progresso desde a graduação. Agradeço ao Prof. Dr. Dalvan Jair Griebler, pela disponibilidade e contribuições como membro externo.

Ao Programa de Pós-Graduação em Ciência da Computação (MDCC). E também a todos os servidores da Universidade Federal do Ceará, em especial aos docentes, cujo trabalho dedicado ao ensino público, gratuito e de qualidade me inspirou.

Aos meus familiares e amigos, pelo apoio incondicional, pela compreensão em meus momentos de ausência e por todo o incentivo.

Aos meus colegas docentes da UFC Campus Quixadá pela amizade e longas conversas sobre as mais diversas áreas da ciência.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

## RESUMO

O presente trabalho aborda o Aprendizado Dividido Federado (ADF), uma técnica que permite o treino de modelos de redes neurais artificiais entre diferentes entidades sem o compartilhamento direto de seus dados, preservando a privacidade. Ao particionar o modelo entre o cliente e o servidor, o ADF também ajuda a reduzir a carga computacional no lado do cliente. Apesar de suas vantagens, utilizar essa solução ainda é um desafio em ambientes computacionalmente limitados, como dispositivos móveis e IoT, especialmente do lado do cliente. Este trabalho analisa o desempenho do uso de quantização, uma técnica que reduz a precisão numérica dos parâmetros e das ativações de redes neurais, para reduzir os custos computacionais no cliente durante a inferência em soluções de ADF. Para isso, um estudo empírico foi conduzido em um ambiente na nuvem, utilizando os modelos VGG11, ResNet18 e MobileNetV2, bem como variações IID e não-IID do conjunto de dados CIFAR-10. Comparam-se os esquemas de Quantização Pós-Treino (QPT) e de Treino Consciente de Quantização (TCQ) com o modelo de precisão completa. Os resultados demonstram que os ganhos e as perdas da quantização estão diretamente ligados à arquitetura, à escolha da camada de corte e à quantização do modelo do cliente. A quantização do modelo do cliente acelerou, em média, a etapa de propagação do cliente em 4,3. A redução no consumo de memória esteve entre 26% a 56%, enquanto a redução do consumo de rede se manteve em 75% para todos os casos. A perda na acurácia também está ligada à arquitetura: enquanto os modelos VGG11 e ResNet18 mantiveram a perda de acurácia menor que 2%, a QPT na arquitetura MobileNetV2 obteve perdas de acurácia de até 22,7%, tornando o uso de TCQ essencial.



**Palavras-chave:** aprendizado profundo; aprendizado dividido federado; quantização.

## ABSTRACT

This paper addresses Split Federated Learning (SFL), a technique that enables artificial neural network models to be trained across multiple entities without directly sharing their data, thereby preserving data privacy. By partitioning the model between the client and the server, SFL also helps to reduce the computational load on the client side. However, despite its advantages, using this solution remains challenging in computationally constrained environments, such as mobile devices and IoT devices, especially on the client side. This work analyzes the performance of quantization, a technique that reduces the numerical precision of neural network parameters and activations to lower computational costs for clients during inference in SFL solutions. To this end, an empirical study was conducted in a cloud environment using the VGG11, ResNet18, and MobileNetV2 models over IID and non-IID variations of the CIFAR-10 dataset. The Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) schemes are compared with the full-precision model. The results demonstrate that the gains and losses from quantization are directly linked to the architecture, the choice of the cut layer, and the data distribution. Quantizing the client model accelerated the client's forward pass by 4.3 on average. The reduction in memory consumption ranged from 26% to 56%, while the decrease in network consumption remained at 75% for all cases. The accuracy loss is also linked to the architecture: while the VGG11 and ResNet18 models maintained an accuracy loss of less than 2%, PTQ on the MobileNetV2 architecture led to losses of up to 22.7%, underscoring the need for QAT.



**Keywords:** deep learning; split federated Learning; quantization.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AD | Aprendizado Dividido |
| ADF | Aprendizado Dividido Federado |
| AF | Aprendizado Federado |
| CPU | Central Processing Unit |
| GD | Gradiente descendente |
| GPU | Graphic Processing Unit |
| gRPC | Google Remote Procedure Call |
| IaaS | Infrastructure as a Service |
| IID | Independente e Identicamente Distribuídas |
| IoT | Internet das Coisas |
| IoV | Internet of Vehicles |
| IR | Resultado Intermediário |
| LGPD | Lei Geral de Proteção de Dados |
| ML | Machine Learning |
| QPT | Quantização Pós Treino |
| RNC | Rede Neural Convolucional |
| SGD | Gradiente descendente estocástico |
| TCQ | Treino Consciente de Quantização |
| TPU | Tensor Processing Unit |
| VGG | Visual Geometry Group |

**SUMÁRIO**

# 1 INTRODUÇÃO

O aprendizado profundo é uma subárea da inteligência artificial focada no estudo de redes neurais artificiais. Essas redes são estruturas computacionais baseadas no processo cognitivo humano, projetadas para analisar grandes volumes de dados e identificar, de forma automatizada, padrões complexos (GOODFELLOW *et al.*, 2016). Atualmente, as redes neurais são consideradas soluções de estado da arte para problemas em diversas áreas da ciência, como a classificação de imagens, a detecção de objetos, a tradução automática de textos e o reconhecimento de voz, dentre outros (ALOM *et al.*, 2019).

Também é reconhecido que soluções de aprendizado profundo precisam de grandes quantidades de dados de treinamento para alcançar boa acurácia (GOODFELLOW *et al.*, 2016). Na prática, o desempenho dessas soluções exige a integração de conjuntos de dados provenientes de diferentes fontes, como hospitais, bancos de dados e dispositivos móveis. Porém, isso implica riscos de vazamento de informações durante a transferência de dados da máquina local para o servidor central, além de risco de um servidor malicioso com acesso direto aos dados (KAIROUZ *et al.*, 2021). Esse problema é ainda mais crítico em cenários que lidam com dados sensíveis ou protegidos por lei (MULHOLLAND, 2018).

Nesse cenário, surgiu o aprendizado colaborativo, um conjunto de técnicas de treino e inferência de modelos em que as entidades não precisam compartilhar seus dados brutos (TRUONG *et al.*, 2021). A primeira e mais popular abordagem desenvolvida foi o Aprendizado Federado (AF), onde os envolvidos no treino compartilham seus modelos em vez de seus dados. Essa abordagem exige que cada cliente armazene e treine o modelo por completo, o que torna a solução inviável em dispositivos computacionais limitados.

Para atacar ambos os problemas de privacidade e de limitação computacional, surgiram o Aprendizado Dividido (AD) e o Aprendizado Dividido Federado (ADF), que particionam um modelo de rede neural em dois submodelos: o do cliente e o do servidor. Essa divisão evita que os dados brutos saiam do cliente e aloca a maior parte do processamento ao servidor.

Porém, mesmo quando o modelo do cliente é apenas uma pequena fração do modelo completo em soluções de ADF, ele ainda demanda alto custo computacional para realizar inferência, principalmente em cenários que envolvem restrições de hardware, tais como IoT e dispositivos móveis (HAFI *et al.*, 2024). Nesse contexto, a técnica de *quantização* surge como uma alternativa para reduzir de custos computacionais da inferência no lado do cliente em redes neurais, no cenário de ADF.

A quantização é uma técnica que reduz a precisão numérica de parâmetros e ativações, com o propósito de diminuir o tamanho do modelo e o custo computacional das operações aritméticas (NAGEL *et al.*, 2021). Valores numéricos representados como pontos flutuantes de 32 bits são convertidos para formatos mais compactos, como inteiros de 8 bits. A quantização reduz o consumo de memória durante a inferência, o tamanho do modelo, os custos de transferência de dados no ADF e o tempo de inferência, uma vez que operações aritméticas com inteiros tendem a ser mais amigáveis ao *hardware*. O desafio da área é alcançar esses benefícios sem comprometer demais a precisão do modelo.

Este trabalho analisa a aplicação da quantização no modelo do cliente em soluções de ADF com o objetivo de reduzir os requisitos computacionais exigidos na operação de inferência no cliente. Para validar essa técnica, um arcabouço computacional foi construído para execução de experimentos com redes neurais convolucionais em um cenário de ADF. Nos experimentos, o servidor foi executado em uma máquina local, enquanto os clientes foram executados em máquinas virtuais com recursos limitados, provisionadas na infraestrutura na nuvem.

Foi realizada a divisão dos modelos de rede neurais entre o modelo do servidor e o modelo dos clientes e, posteriormente, o seu treinamento. As arquiteturas utilizadas nos experimentos foram as redes neurais convolucionais *VGG11*, *ResNet18* e *MobileNetV2*. Também foram utilizados dois esquemas de quantização: Quantização Pós Treino (QPT) e Treino Consciente de Quantização (TCQ). Como resultado, foram geradas e comparadas três versões dos modelos: uma com precisão completa, usada como *baseline*, e outras duas versões quantizadas com QPT e TCQ. Em relação ao conjunto de dados, foi utilizado o *CIFAR-10* (com IID e não-IID).

Para a análise do custo computacional, foram coletadas métricas de tempo de propagação do cliente, pico de memória no cliente, tamanho do modelo do cliente, consumo de rede e acurácia. Além disso, foi realizada uma análise de acurácia, permitindo avaliar o equilíbrio entre a redução potencial da acurácia do modelo e a redução dos requisitos computacionais.

## 1.1 Objetivo

O objetivo geral deste trabalho é avaliar o impacto do uso de quantização no modelo do cliente nas operações de inferência em soluções de Aprendizado Dividido Federado (ADF).

### *1.1.1 Objetivos específicos*

Os objetivos específicos desse trabalho são:

- Avaliar como a quantização do modelo do cliente pode reduzir os custos computacionais relacionados à inferência no lado do cliente.
- Avaliar o impacto na acurácia do modelo global decorrente da redução de precisão numérica causada pela quantização do modelo do cliente.
- Avaliar o equilíbrio entre ganho computacional e perda de acurácia na quantização do modelo do cliente.
- Desenvolver recomendações para o uso de quantização no ADF com base na análise dos resultados dos experimentos.

## 1.2 Organização do documento

O restante deste trabalho está organizado em mais quatro capítulos. No Capítulo 2 são abordados os conceitos fundamentais para a compreensão da pesquisa, incluindo aprendizado de máquina, aprendizado colaborativo e quantização de redes neurais, além da discussão de trabalhos relacionados. Em seguida, o Capítulo 3 descreve as configurações dos experimentos realizados com detalhes dos conjuntos de dados, dos modelos, da divisão dos modelos, do treino, dos esquemas de quantização, do ambiente computacional e das métricas coletadas. Por sua vez, o Capítulo 4 apresenta e discute os resultados obtidos nos experimentos através de uma análise de custo computacional e uma análise de acurácia. Finalmente, o Capítulo 5 apresenta considerações finais, resumindo as principais contribuições, as limitações da pesquisa e as sugestões para trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Este capítulo apresenta os principais conceitos necessários à compreensão do presente trabalho. A Seção 2.1 apresenta conceitos fundamentais de inteligência artificial, incluindo aprendizado de máquina, aprendizado supervisionado e aprendizado profundo. Em seguida, a Seção 2.2 aborda três soluções predominantes para o aprendizado colaborativo: Aprendizado Federado (AF), Aprendizado Dividido (AD) e a solução mista Aprendizado Dividido Federado (ADF). A Seção 2.6 aborda a técnica de quantização, avaliada neste trabalho para aprimorar o desempenho do ADF. Por fim, a Seção 2.7 discute pesquisas relacionadas ao presente trabalho.

## 2.1 Aprendizado de máquina

Aprendizado de máquina é uma subárea da inteligência artificial que, de acordo com Burkov (2019), pode ser definida como conjunto de métodos para solucionar um problema através da coleta de dados e construção algorítmica de um modelo estatístico baseado nesses dados coletados. Em outras palavras, busca-se implementar um sistema capaz de extrair, de forma automatizada, informações úteis de um conjunto de dados por meio de modelos estatísticos.

O processo de aprendizagem difere de acordo com o tipo de dado que é fornecido para o modelo (BURKOV, 2019). Dependendo do formato do problema e do conjunto de dados, pode-se categorizar a aprendizagem em três paradigmas:

- Aprendizado supervisionado: O conjunto de dados é uma coleção de instâncias rotuladas. Para cada instância do conjunto de dados, temos um rótulo que o caracteriza. O desafio da área é, dado uma instância, inferir o valor do rótulo ainda não observado. Problemas de classificação e regressão são exemplos dessa categoria.
- Aprendizado não supervisionado: O conjunto de dados não está rotulado. Dada uma entrada, o modelo não tem conhecimento da saída esperada. Nesse caso, o modelo deve se adequar naturalmente a padrões ocultos nos dados. A clusterização é um exemplo de problema nessa categoria.
- Aprendizado por reforço: O modelo não possui acesso a um conjunto de dados, mas é capaz de perceber o ambiente a partir de seu estado. O modelo pode se mover entre os estados por meio de ações recompensadas por um agente externo. O valor da recompensa deve ser proporcional ao benefício trazido pela ação. Dessa forma, o modelo aprende a tomar decisões que maximizem o valor esperado de recompensas.

### *2.1.1 Aprendizado supervisionado*

No aprendizado supervisionado, o modelo deve mapear cada valor de entrada para um determinado valor de saída utilizando um conjunto de dados de treino como exemplo (BURKOV, 2019). O conjunto de dados pode ser descrito como uma coleção $\{(x_i, y_i)\}_1^N$, onde $x_i$ é um vetor de atributos que descrevem a instância $i$ e $y_i$ é um valor de rótulo que está associado a instância $i$. O valor de um rótulo pode ser um elemento de um conjunto finito de classes (problema de classificação), um número real (problema de regressão) ou alguma estrutura de dados mais complexa como, por exemplo, um vetor, matriz ou grafo.

Neste trabalho, focamos no problema de classificação de imagens, um problema onde o modelo recebe uma imagem, geralmente representada por uma matriz numérica, e retorna um rótulo que indica a qual classe essa imagem pertence. Geralmente, essa classe é expressada como um nome ou número inteiro. Um exemplo comum desse tipo de tarefa é o conjunto de dados CIFAR-10 (abordado na Seção 3.3), que consiste em pequenas imagens coloridas anotadas com classes como “avião”, “automóvel”, “pássaro” e “gato”. Através desses dados rotulados, o modelo precisa predizer a classe de dados ainda não vistos de forma acurada com base em seus padrões visuais.

#### *2.1.1.1 Tipos de conjuntos de dados*

Na prática, o conjunto completo de todos os dados do problema não está disponível para o modelo. Em vez disso, se trabalha com um subconjunto de dados limitado, que é geralmente dividido em três partes (BURKOV, 2019):

- Conjunto de dados de treino: utilizado pelo algoritmo de treino na etapa de aprendizagem para otimizar os parâmetros do modelo.
- Conjunto de dados de validação: usado para ajustar parâmetros que não podem ser otimizados durante o treino, ajudando na seleção e na avaliação de hiperparâmetros.
- Conjunto de dados de teste: utilizado para medir a qualidade final do modelo, simulando seu desempenho em dados não vistos durante o treino.

Essa divisão dos conjuntos é fundamental para criar um ambiente de aprendizagem mais realista, onde o modelo é validado e testado com instâncias não conhecidas previamente (GOODFELLOW *et al.*, 2016). Sua principal função é evitar sobreajuste (*overfitting*) do modelo, ou seja, garantir que o modelo funcione bem para novos dados não observados, ao invés de

funcionar apenas para os dados do conjunto de treino.

#### *2.1.1.2 Função de perda*

Um passo importante para a aprendizagem de máquina é mensurar a qualidade das predições de um modelo. A função de perda serve para medir essa qualidade, calculando a distância entre a saída do modelo e o rótulo real dos dados de treino (GOODFELLOW *et al.*, 2016). O objetivo do treino de um modelo é minimizar a função de perda e, dessa forma, melhorar a precisão das predições.

Um exemplo de função de perda utilizada para problemas de regressão inclui o erro quadrático médio representada na Equação 2.1, onde $\hat{y}_i$ é a saída predita do modelo para a instância $i$ e $N$ é o tamanho do conjunto de dados.

$$MSE(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.1}$$

Para classificação, um exemplo comum de função de erro é a Entropia Cruzada descrita na Equação 2.2, onde $N$ é o tamanho do conjunto de dados, $M$ é o número total de classes, $y_{i,c}$ é igual a 1 se a instância $i$ pertencer à classe $c$ ou 0 caso contrário, $\hat{y}_{i,c}$ é a probabilidade predita para a instância $i$ pertencer a classe $c$.

$$CE(y,\hat{y}) = -\sum_{i=1}^{N}\sum_{c=1}^{M} y_{i,c}\log(\hat{y}_{i,c}) \tag{2.2}$$

### *2.1.2 Aprendizado profundo*

Aprendizado profundo é uma subárea de aprendizado de máquina que se baseia no estudo de redes neurais profundas para o reconhecimento e representação de padrões de dados (CHOLLET, 2021). O termo "profundo" se refere ao aprendizado que é realizado através de neurônios artificiais que são organizados em camadas consecutivas de representações abstratas.

Soluções de aprendizado profundo têm se destacado em várias áreas da ciência, sendo capaz de alcançar níveis de inteligência humana para conclusão de tarefas (CHOLLET, 2021). Dentre essas tarefas, se destacam classificação de imagens, transcrição de voz, tradução de textos e simulação de conversas (*chatbots*). As soluções de aprendizado profundo revolucionaram esses domínios ao permitir que os modelos aprendessem automaticamente representações

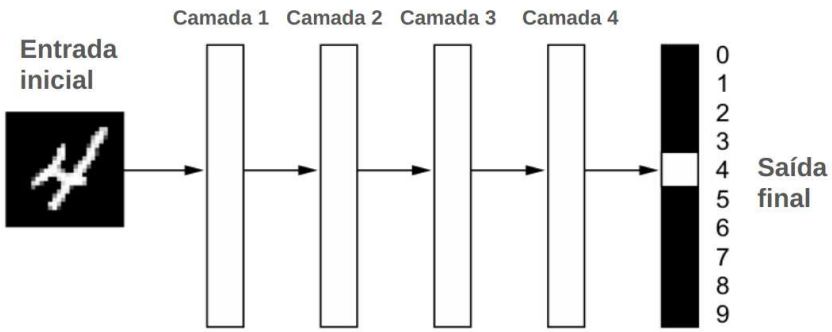
mais complexas e hierárquicas dos dados, reduzindo a necessidade de extração manual de características.

#### *2.1.2.1 Redes neurais profundas*

Uma rede neural profunda é uma estrutura que se baseia no comportamento cognitivo do cérebro humano através de funções matemáticas simples interconectadas denominadas neurônios artificiais (HAYKIN, 2009). Esses neurônios se comunicam enviando e recebendo estímulos entre si. Um neurônio é ativado quando recebe sinais externos (entrada) ou recebe um sinal de um outro neurônio previamente ativado e, após sua ativação, o mesmo envia um sinal de saída para outros neurônios ou para o ambiente externo (saída).

O termo profundo se refere a arquitetura das redes neurais, que normalmente é organizada em camadas sequenciais, onde cada camada utiliza os sinais de saída da camada anterior para gerar um novo sinal (CHOLLET, 2021). Quanto maior o número de camadas de um modelo, mais profundo ele é.

Figura 1 – Exemplo da estrutura de uma rede neural que recebe como entrada uma imagem e retorna qual o dígito correspondente.




Fonte: Adaptado de Chollet (2021)

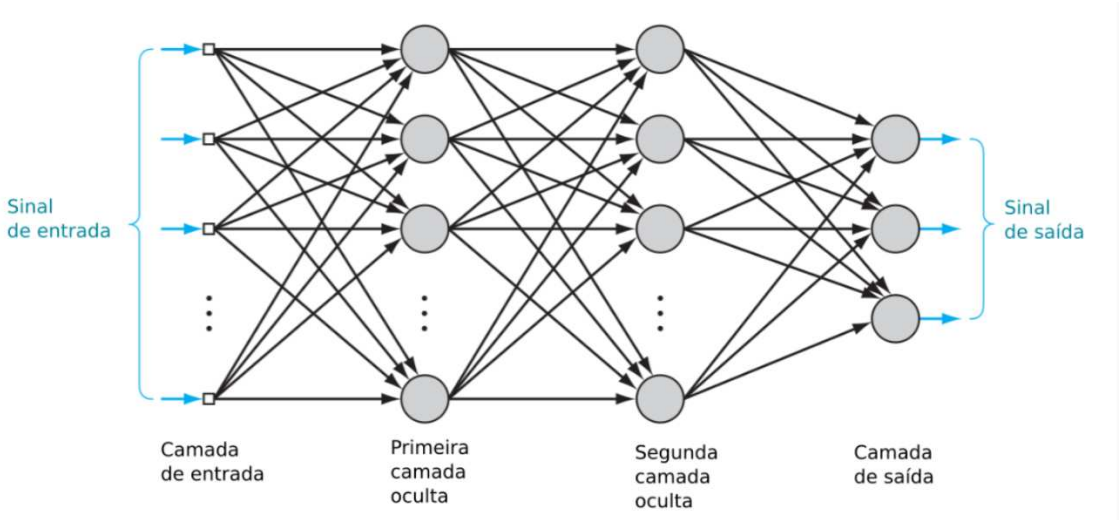
Um exemplo de arquitetura de rede neural é ilustrada na Figura 2. As camadas dessa uma rede neural são divididas em três tipos, dependendo de suas funções:

- Camada de entrada: consiste nos neurônios da camada inicial que são alimentados diretamente com os atributos da instâncias do problema a ser resolvido. Sua quantidade de neurônios depende do número de atributos do dado de entrada.
- Camadas ocultas: são as camadas intermediárias, cujo o papel é aplicar transformações não-lineares nos dados obtidos da camada de entrada. Fatores como complexidade do

problema, capacidade computacional e generalização devem ser considerados na decisão do formato das camadas ocultas.

- Camada de saída: é a camada final responsável por representar o resultado da rede neural após processar a saída das camadas ocultas. Sua quantidade de neurônios depende do formato da saída desejada.

Figura 2 – Exemplo de estrutura de uma rede neural profunda.




Fonte: Adaptado de Chollet (2021)

### *2.1.2.2 Redes neurais convolucionais*

Uma Rede Neural Convolucional (RNC) é um tipo específico de rede neural profunda especializada no processamento de dados com formato de matriz como, por exemplo, uma imagem (matriz bidimensional ou tridimensional) ou uma série temporal (matriz unidimensional) (GOODFELLOW *et al.*, 2016). Atualmente, RNCs são utilizadas de forma universal em tarefas de visão computacional como classificação de imagens, detecção de objetos e segmentação.

As RNCs são caracterizadas por suas camadas convolucionais, responsáveis por utilizar filtros para aplicar operações de convolução na entrada (GOODFELLOW *et al.*, 2016). Um filtro é uma pequena matriz de pesos que percorre a entrada em diferentes posições, multiplicando e somando seus valores com a saída da camada anterior para produzir uma outra saída denominada mapa de características (*feature maps*, em inglês). Esse resultado representa características extraídas como, por exemplo, bordas e texturas de uma imagem.

Normalmente, uma RNC começa com uma sequência de camadas convolucionais junto a operações não-lineares, responsáveis por reconhecer padrões espaciais. Essas camadas

são seguidas por operações de agrupamento (*pooling*, em inglês), que reduzem o tamanho das dimensões dos mapas de características para melhorar a eficiência. Após essa sequência de extração de características, a rede costuma incluir camadas densas que processam os mapas de características extraídos e geram a saída final do modelo (GOODFELLOW *et al.*, 2016).

## 2.2 Aprendizado colaborativo

Com o crescimento do impacto de soluções de aprendizado de máquina e o surgimento de leis de proteção de dados, garantir a privacidade dos dados tem-se tornado um requisito para o avanço da tecnologia (TRUONG *et al.*, 2021). Por exemplo, a Lei Geral de Proteção de Dados (LGPD) no Brasil estabelece diretrizes para o tratamento de dados pessoais, especialmente aqueles considerados sensíveis, como informações sobre saúde, raça e orientação sexual (MULHOLLAND, 2018). Nesse contexto, a aprendizagem de máquina descentralizada surgiu como uma solução para permitir que várias entidades participem do treino de modelos de aprendizado de máquina sem infringir as regulações de segurança e privacidade.

Aprendizado de máquina colaborativo é um paradigma onde o treinamento do modelos é distribuído entre várias entidades (LUDWIG; BARACALDO, 2022). Ele diferente da abordagem tradicional centralizada, onde os dados são reunidos em um servidor central. Essas entidades que participam do treino mantêm seus dados localmente durante o treino e a inferência, sem compartilhá-los diretamente com os demais participantes ou agentes externos, garantindo um nível de privacidade e segurança.

Como a qualidade de modelos estatísticos depende da quantidade de dados disponíveis, o aprendizado de máquina descentralizado se mostra útil ao tornar possível o treino colaborativo entre várias entidades com dados sensíveis. Exemplos de cenários onde a privacidade dos dados é essencial incluem aplicações hospitalares, Internet of Vehicles (IoV), aplicações financeiras e dispositivos móveis (LI *et al.*, 2020).

Apesar dos benefícios, lidar com uma arquitetura descentralizada traz diversos desafios, dentre os quais se destacam ataques de segurança e privacidade, heterogeneidade de dados, limitações de hardware, comunicação entre os participantes, dentre outros.

## 2.3 Aprendizado federado

O Aprendizado Federado (AF) é uma solução de aprendizado colaborativo inicialmente desenvolvida pela *Google* para o treino de um modelo de predição da próxima palavra do teclado virtual em dispositivos móveis *Android* (MCMAHAN; RAMAGE, 2017). A solução foi necessária devido à sensibilidade dos dados textuais digitados pelo usuário em seus aparelhos.

Em uma solução clássica de AF, os participantes do treino são separados entre clientes e servidor (LUDWIG; BARACALDO, 2022). Os clientes são responsáveis por coletar, armazenar os dados, manter e treinar localmente uma cópia do modelo global. Já o servidor é responsável por reunir os modelos treinados individualmente por cada cliente, aplicar uma função para agregar os modelos e distribuir esse novo modelo global novamente para os clientes.

Para manter a privacidade, os dados sensíveis são mantidos localmente pelo cliente durante todo o processo de treino e inferência. Isso é alcançado pois as únicas informações que o cliente compartilha são parâmetros do modelo treinado localmente, ao invés dos próprios dados, como em uma solução centralizada tradicional. Da mesma forma, as únicas informações compartilhadas pelo servidor são parâmetros do modelo atualizado.

Porém, apesar do AF realizar apenas o compartilhamento de parâmetros do modelo, alguns estudos têm exposto vulnerabilidades que podem surgir a partir desses artefatos gerados no treino (LYU *et al.*, 2022). Além disso, o AF também sofre da desvantagem de que cada cliente é responsável por armazenar e treinar o modelo por completo, tornando a solução inviável em ambientes com *hardware* limitado.

### *2.3.1 Algoritmo de treino no aprendizado federado*

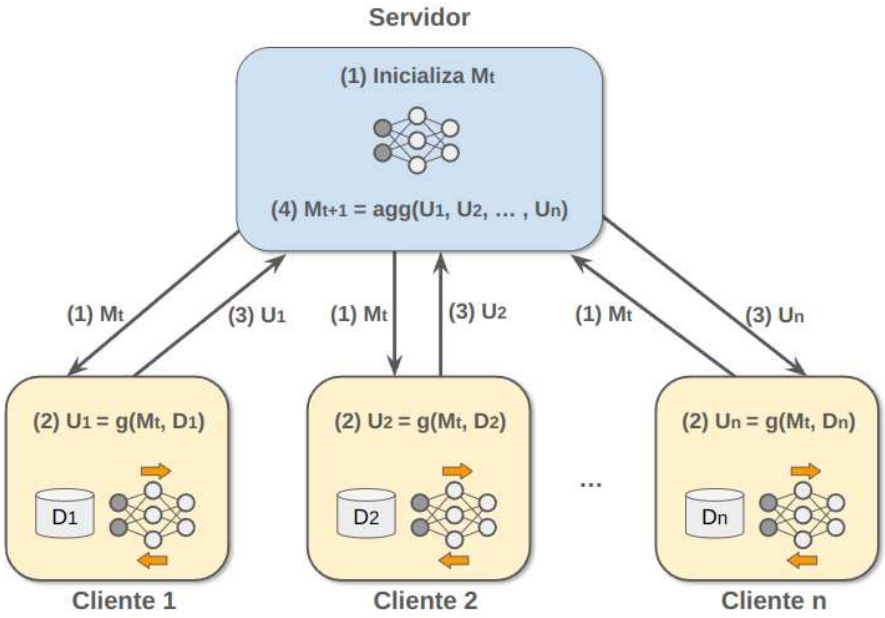
Após inicializar o modelo global, o servidor distribui uma cópia do modelo para cada cliente. Em seguida, o cliente realiza o treino do modelo utilizando seus dados locais, gerando um novo modelo local em cada cliente. Após esse treino local, cada cliente envia os parâmetros de seu modelo para o servidor. O servidor, por sua vez, une todos os modelos locais através de uma função de agregação, gerando um novo modelo global atualizado.

Mais especificamente, o algoritmo do AF realiza o treino de um modelo global $M_t$ que é distribuído entre vários clientes $C = \{C_1, C_2, ..., C_n\}$, onde cada cliente $i$ possui seu próprio conjunto de dados $D_i$ localmente.

O algoritmo é executado de forma iterativa, onde cada iteração é enumerada por $t$. A

seguir, será descrito o passo a passo da implementação original do AF, ilustrada na Figura 3:

Figura 3 – Implementação clássica do AF




Fonte: Elaborado pelo autor.

1. O servidor inicializa o modelo $M_t$. Na primeira iteração, isso pode ser realizada de forma arbitrária ou seguindo algum algoritmo específico (KUMAR, 2017). Nas demais iterações, esse modelo é o resultado da iteração anterior.
2. A primeira rodada de comunicação acontece, onde o servidor compartilha uma cópia completa do modelo $M_t$ para todos os clientes.
3. Após receber a cópia do modelo $M_t$, cada cliente $C_i$ realiza o treino utilizando seus dados locais $D_i$ e uma função de treino $g$. Essa função retorna um novo modelo $U_i$ especializado para os dados cliente $C_i$.
4. A segunda rodada de comunicação ocorre, onde o servidor aguarda receber de volta o modelo gerado de cada um dos clientes.
5. O servidor realiza a operação de agregação através de uma função *agg*, que une os modelos locais dos cliente para gerar um novo modelo global $M_{t+1}$. A função *FedAvg* (SUN *et al.*, 2021) é um exemplo simples de agregação, onde o novo modelo é calculado através da média dos parâmetros de todos os modelos locais.
6. Os passos de $1-5$ são repetidos até que algum critério de parada seja atendido como, por exemplo, convergência alcançada ou tempo limite excedido.

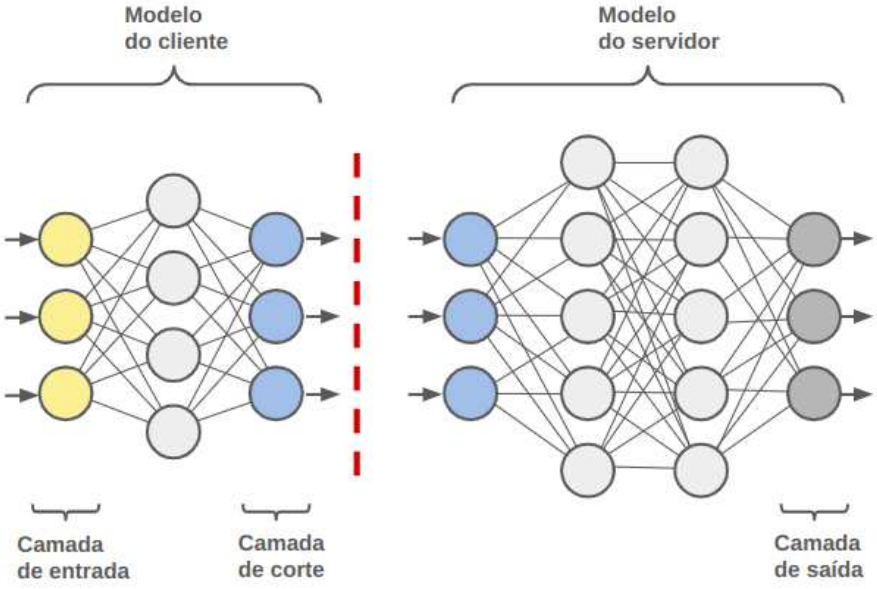
Essa versão básica, inicialmente proposta por McMahan e Ramage (2017). No entanto, a literatura atual já disponibiliza diversas variações em cada um dos passos citados.

## 2.4 Aprendizado dividido

Para obter resultados satisfatórios em soluções de aprendizado profundo, é comum o uso de modelos extremamente complexos, o que exige um poder computacional considerável para o treinamento (MENGHANI, 2023). Essa exigência se torna um grande desafio em cenários de aprendizado colaborativo, onde os dispositivos do usuários envolvidos possuem um poder computacional limitado. Por exemplo, ambientes móveis e IoT apresentam restrições computacionais nos dispositivos.

Tendo em vista esse cenário, o Aprendizado Dividido (AD) surge como uma solução de aprendizado colaborativo, com foco no treino de clientes com *hardware* limitado, sem compartilhar diretamente os dados sensíveis (GUPTA; RASKAR, 2018). Para isso, a arquitetura da rede neural é dividida entre o cliente e o servidor. Essa divisão normalmente é feita por camadas, e o cliente fica encarregado das camadas sensíveis, que têm contato mais direto com os dados, como a camada de entrada e as primeiras camadas convolucionais. Dessa forma, as demais camadas são alocadas no servidor, aproveitando o *hardware* mais robusto. Um exemplo dessa divisão de arquitetura é ilustrado na Figura 4.

Figura 4 – Exemplo de arquitetura de uma rede neural que implementa AD. As primeiras camadas, de entrada à camada de corte, são armazenadas no cliente. As demais são armazenadas no servidor.




Fonte: Elaborado pelo autor.

No exemplo da Figura 4, o modelo é particionado em dois submodelos menores: o modelo do cliente e o modelo do servidor. Para se realizar uma inferência, o cliente aplica

a operação de propagação sobre seus dados no seu modelo. A saída do modelo do cliente é enviada ao servidor, onde servirá de entrada para o modelo do servidor, que, por sua vez, gera a saída final do modelo. A operação de retropropagação é realizada de forma semelhante, porém no sentido inverso.

Dessa forma, não é necessário que o cliente compartilhe diretamente seus dados. Em vez disso, ele compartilha o resultado das operações aritméticas que o modelo do cliente aplica sobre a entrada. Isso garante um certo nível de segurança, mas ainda é possível inferir propriedades dos dados a partir dessa saída (PASQUINI *et al.*, 2021). Além disso, a implementação do AD permite reduzir drasticamente a carga computacional no lado do cliente, já que requer acesso apenas a uma fração do modelo global.

### *2.4.1 Algoritmo de treino no aprendizado dividido*

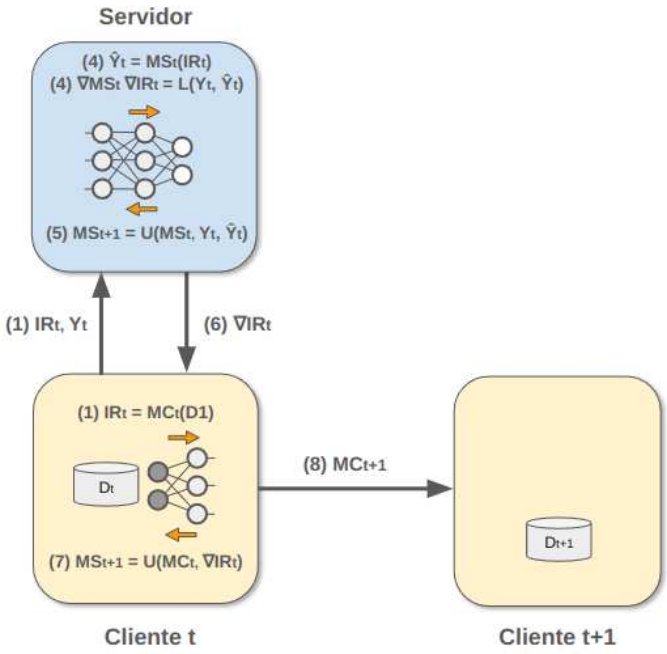
Na implementação tradicional do AD, em sua versão oficialmente proposta por Gupta e Raskar (2018), o objetivo é realizar o treino de um modelo global que é dividido entre um modelo do cliente $MC$ e um modelo do servidor $MS$. O cliente mantém todas as camadas do modelo desde a camada de entrada até uma certa camada denominada camada de corte. As camadas posteriores à de corte são armazenadas pelo servidor.

Nessa versão, a cada iteração $t$, o treino é realizado de forma sequencial entre o servidor e o cliente $C_t$, utilizando apenas o próprio conjunto de dados local $D_t$, enquanto os demais clientes aguardam por sua vez. No final da iteração $t$, o cliente $C_t$ envia a nova versão do modelo $MC_{t+1}$ ao próximo cliente $C_{t+1}$, para que ele continue o treino com novos dados locais.

A seguir, a solução será descrita com mais detalhes, conforme ilustrado na Figura 5:

1. O modelo do cliente $MC_t$ é alimentado com seu conjunto de dados locais, gerando um resultado intermediário $IR_t$. O $IR_t$ é o resultado das operações do modelo do cliente desde sua camada de entrada até a camada de corte;
2. O $IR_t$ é enviado do cliente $C_t$ para o servidor para que o treino prossiga. No caso do aprendizado supervisionado, é necessário também o envio do rótulo $Y_t$;
3. O modelo do servidor $MS_t$ é alimentado com o $IR_t$, gerando uma predição $\hat{Y}$. Logo após, o servidor realiza a operação de retropropagação, onde são calculados $\nabla MS_t$ e $\nabla MS_t$, que são, respectivamente, o gradiente dos parâmetros do modelo $MS_t$ e o gradiente da entrada $IR_t$ em relação à função de erro $L(Y,\hat{Y})$;
4. O modelo do servidor é atualizado através de uma função de atualização $U$ e o gradiente

Figura 5 – Passo a passo do treinamento de uma solução AD




Fonte: Elaborado pelo autor.

$\nabla MS_t$;

5. O $\nabla IR_t$ é enviado do servidor para o cliente, para que o mesmo seja atualizado;
6. O cliente utiliza o gradiente do seu resultado intermediário $\nabla IR_t$ para continuar o processo de retropropagação em seu modelo, calculando $\nabla MC_t$.
7. O modelo do cliente é atualizado a partir e seu gradiente $\nabla MC_t$;
8. O cliente $C_t$ compartilha o novo modelo atualizado $MC_{t+1}$ com o próximo cliente $C_{t+1}$ para a próxima rodada.
9. Os passos de $1-8$ são repetidos até que algum critério de parada seja atendido como, por exemplo, convergência alcançada ou tempo limite excedido.

Essa versão básica exemplifica o funcionamento do AD. Uma de suas principais limitações de privacidade inclui o compartilhamento do Resultado Intermediário (IR) que pode ser mais suscetível a ataques de inferência em relação ao compartilhamento de parâmetros, o compartilhamento do rótulo $Y_t$ e o algoritmo sequencial, onde apenas um cliente realiza o treino por rodada, enquanto os demais permanecem ociosos. As próximas seções exploram essas limitações e abordam variações do AD que mitigam esses problemas.

### *2.4.2 Segurança e privacidade*

Manter a privacidade sobre os dados é uma das principais motivações para o uso de técnicas de aprendizado descentralizado como o AD ou AF (LI *et al.*, 2020). O AD busca alcançar a anonimidade ao compartilhar somente o resultado das camadas intermediárias gerados durante o treino, evitando que dados locais sejam diretamente acessados por demais entidades além do próprio proprietário. Ainda assim, diversos estudos apontam vulnerabilidades de segurança e privacidade advindas da técnica.

### *2.4.3 Processamento e memória*

No AF, como cada cliente armazena e treina uma cópia completa do modelo, o aprendizado pode exigir grande capacidade computacional do cliente em questões de processamento, memória e rede (LUDWIG; BARACALDO, 2022). Ao mesmo tempo, esses requisitos muitas vezes não estão disponíveis em ambientes de *hardware* limitado, como *mobile* e Internet das Coisas (IoT). Por consequência, projetistas são obrigados a reduzir a complexidade do modelo ou limitar a quantidade de clientes participantes no treino, reduzindo a qualidade da solução.

O AD surgiu com o propósito de atender a esse tipo de ambiente computacional (GUPTA; RASKAR, 2018), permitindo transferir grande parte dos requisitos de processamento e memória para uma máquina servidora, que normalmente possui um *hardware* mais robusto e escalável. Dessa forma, o AD permite que o modelo seja parcialmente treinado com um *hardware* especializado, como GPUs ou TPUs, os quais, em muitos casos, não são acessíveis pelo cliente.

No AD, a carga de processamento e memória do cliente depende da forma em que o modelo é dividido entre o servidor e o cliente. Por exemplo, ao se implementar o AD original, a carga depende da escolha da camada de corte: quanto mais distante da camada de entrada, maior será o modelo do cliente e, consequentemente, maior o custo de processamento e memória para o cliente. Ao mesmo tempo, uma camada de corte muito próxima à camada de entrada torna o modelo mais suscetível a ataques de inferência, uma vez que o dado original sofre uma quantidade menor de transformações. Essa relação entre custo computacional e privacidade deve ser levada em consideração no projeto da solução.

### *2.4.4 Comunicação*

O custo de comunicação no AD consiste no compartilhamento do IR gerado após a camada de corte e o modelo do cliente atualizado. Diferente do AF que compartilha o modelo por completo, o AD consegue reduzir drasticamente a carga de dados transferidos ao transmitir apenas o modelo completo (LUDWIG; BARACALDO, 2022). Porém, o envio do IR domina o custo esse custo de comunicação, que é calculado de acordo com as dimensões do próprio IR gerado após a camada de corte (GAO *et al.*, 2020).

Esse custo de comunicação no AD é um gargalo devido às limitações de rede, em especial a latência de rede. Uma das técnicas mais propostas para mitigar o custo de comunicação é a redução do tamanho do IR. Exemplos de técnicas incluem seleção de camadas de corte de baixa dimensão, quantização, *autoencoders* e compactação.

### *2.4.5 Sequencial vs Paralelo*

Um dos principais gargalos do AD é seu algoritmo sequencial, onde o servidor realiza o treino com um cliente por iteração (THAPA *et al.*, 2022). Em um cenário com um grande número de clientes, isso aumenta significativamente a sobrecarga do algoritmo, uma vez que grande parte dos clientes permanece ociosa durante o treino.

Além do impacto no desempenho computacional, a abordagem sequencial também agrava problemas de dados não IID. Isso ocorre porque os dados utilizados em cada lote são obtidos através de fontes diferentes, ou seja, os passos do Gradiente descendente estocástico (SGD) são realizados por meio de dados enviesados. Por consequência, o modelo pode convergir mais lentamente ou até mesmo não alcançar a convergência.

### *2.4.6 Qualidade do modelo*

O efeito que utilizar o AD tem sobre a qualidade final de um modelo ainda é um tópico em estudo (GAO *et al.*, 2020). As principais questões envolvem se o modelo converge, o quão rápido ele converge e qual a acurácia do modelo após a conversão.

Isso ocorre pois, apesar da implementação clássica do AD seguir um algoritmo semelhante ao Gradiente descendente (GD), novos desafios surgem em um cenário de treino descentralizado. Dispositivos distintos tendem a gerar ou coletar dados não-IID.

Porém, o algoritmo do gradiente descendente, assim como grande parte dos algorit-

mos de aprendizado de máquina, pressupõe que os dados são distribuídos de forma IID. Esse conflito pode aumentar drasticamente a complexidade de uma solução AD. Em casos extremos, o modelo sofre grande perda na acurácia e pode não alcançar a convergência.

De acordo com o estudo realizado por Gao *et al.* (2020), experimentos indicam que AD obtém modelos com maior acurácia e menor tempo de treino em comparação ao mesmo modelo treinado com AF em dados IID ou desbalanceados. Porém, ao se utilizar um ambiente de dados extremamente não-IID, o modelo com AD não obtém a convergência.

Se torna responsabilidade do projetista aplicar técnicas para mitigar os efeitos de dados não-IID. Exemplos de soluções para lidar com dados heterogêneos incluem:

1. Personalização do modelo: após o treino de um modelo global, cada dispositivo pode realizar individualmente uma nova bateria de treino utilizando somente seus dados locais (TAN *et al.*, 2023; WADHWA *et al.*, 2023). Dessa forma, o modelo pode se tornar mais especializado para os dados locais do dispositivo;
2. Paralelização: a natureza sequencial do AD faz com que cada atualização do modelo seja realizada com dados de um único cliente, possivelmente retardando a convergência (THAPA *et al.*, 2022). Paralelizar a solução permite utilizar os dados de múltiplos clientes em cada passo.

## 2.5 Aprendizado dividido federado

Uma das principais limitações do AD é seu treino sequencial, onde cada cliente processa seus dados locais um por vez. Estudos indicam que isso não apenas desperdiça recursos computacionais, pois deixa a maior parte dos clientes ociosos durante o treinamento (THAPA *et al.*, 2022), como também leva à degradação na qualidade final do modelo, especialmente em casos onde os dados são extremamente desbalanceados (GAO *et al.*, 2020).

Pensando nessas limitações, o Aprendizado Dividido Federado (ADF) surgiu como um paradigma de aprendizado colaborativo, inicialmente proposto por Thapa *et al.* (2022), onde o princípio de agregação do AF é utilizado para permitir o paralelismo no AD.

O paradigma ADF é semelhante ao AD, onde o modelo total é dividido em um modelo do cliente e um modelo do servidor, porém sua principal diferença está na forma em que o treino é realizado. No ADF, os clientes participam do treino de forma paralela e têm seus modelos sincronizados por uma função de agregação.

### *2.5.1 Algoritmo de treino no aprendizado dividido federado*

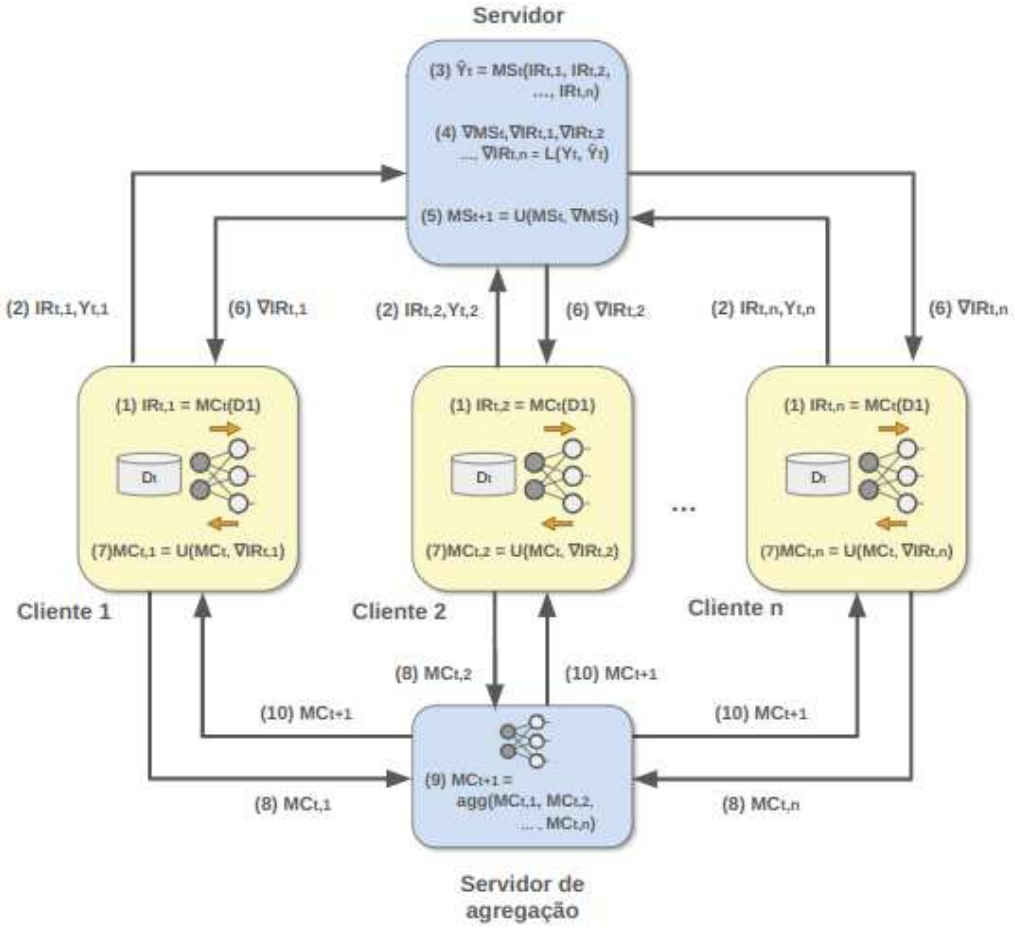
A etapa de treino do ADF é similar ao AD, mas ao invés de realizar a propagação de dados com um cliente por vez, todos os clientes realizam essa etapa simultaneamente, propagando seus dados locais e enviando o IR para o servidor. Em seguida, o servidor une todos os IRs, conclui a etapa de propagação e inicia a retropropagação, retornando um vetor de gradientes para cada cliente. O modelo do servidor é atualizado com o gradiente dos erros provenientes dos dados de todos os clientes, enquanto o modelo de cada cliente é ajustado com o gradiente do erro de seus próprios dados locais. Esse processo gera um modelo diferente para cada um dos clientes, formando um conflito entre os clientes. Para unificá-los, assim como no aprendizado federado, é realizada a operação de agregação dos modelos dos clientes através de um servidor de agregação Thapa *et al.* (2022).

Por exemplo, na implementação tradicional do ADF, o modelo do cliente mantém todas as camadas do modelo total desde a camada de entrada até a camada de corte. As camadas posteriores à camada de corte são armazenadas pelo servidor.

O treino é realizado de forma iterativa, onde cada iteração é enumerada por $t$. A seguir será descrito o algoritmo inicialmente proposto por (THAPA *et al.*, 2022), ilustrado de forma resumida na Figura 6.

1. Em paralelo, cada cliente $C_i$ aplica a propagação de um lote de seus dados locais $D_i$ em seu modelo $MC_t$, gerando um resultado intermediário $IR_i$.
2. Ainda em paralelo, os clientes enviam seus resultados intermediários $IR_i$ para o servidor, junto aos rótulos dos dados $Y_i$.
3. O servidor realiza a propagação dos $IR$s de todos os clientes em seu modelo do servidor $MS_t$, gerando uma predição $\hat{Y}$. Em seguida, o servidor realiza a operação de retropropagação, onde são calculados o gradiente dos parâmetros do modelo do servidor $\nabla MS_t$ e os gradientes dos resultados intermediários dos clientes $\nabla IR_1, \nabla IR_2, ..., \nabla IR_n$ em relação a função de perda.
4. O modelo do servidor é atualizado através do $\nabla MS_t$.
5. Para continuar a retropropagação, o servidor envia os gradientes dos resultados intermediários - $\nabla IR_1, \nabla IR_2, ..., \nabla IR_n$ - para cada um dos respectivos clientes.
6. Cada cliente $C_i$, em paralelo, continua a retropropagação utilizando $\nabla IR_i$ para calcular o gradiente dos parâmetros de seu modelo em relação à função de perda e atualiza seu modelo $MC_{t,i}$.

Figura 6 – Ilustração do algoritmo de treino do ADF.




Fonte: Elaborado pelo autor.

7. Para evitar que os modelos dos clientes sejam diferentes entre si, os modelos dos clientes $MC_{t,i}$ são enviados para o servidor de agregação.
8. O servidor de agregação aplica uma função de agregação *agg* sobre os modelos dos clientes, gerando um novo modelo $MC_{t+1}$ que reúne o conhecimento de todos os clientes.
9. O servidor envia o novo modelo $MC_{t+1}$ para todos os clientes.
10. Os passos de 1 - 9 são repetidos até a solução alcançar algum critério de parada como, por exemplo, atingir a acurácia desejada.

### *2.5.2 Inferência no aprendizado dividido federado*

A inferência em soluções de ADF ocorre de forma idêntica ao AD, ou seja, o cliente propaga seus dados locais através do modelo do cliente, gerando um IR. Esse IR é então enviado do cliente para o servidor, que continua com a propagação do IR em seu modelo do servidor, gerando o rótulo predito.

### *2.5.3 Segurança e privacidade*

Durante o treino no ADF, a privacidade sobre os dados é mantida através do compartilhamento de apenas o resultado intermediário do modelo do cliente e os parâmetros do modelo do cliente, ao invés de compartilhar diretamente os dados locais com o servidor.

Com isso, o ADF herda não apenas a segurança e privacidade do AD e aprendizado federado, como também suas fragilidades. Ataques de inferência podem ser aplicados em ambos IR e parâmetros do modelo do cliente compartilhados durante a agregação, tornando-se necessário defender ambos os artefatos.

### *2.5.4 Processamento e memória*

O processamento realizado no treino de uma solução de ADF depende da forma em que o modelo é dividido e da forma em que os modelos dos clientes são agregados.

A divisão do modelo do cliente e modelo do servidor afeta as operações de propagação e retropropagação no treino e permite alocar a maior parte da carga de processamento para o servidor. Além disso, diferente do AD tradicional, é possível que todos os clientes (ou um grupo deles) participem de cada iteração de forma paralela, evitando a ociosidade das máquinas dos clientes durante o treino, mas aumentando a carga nos clientes.

Além disso, o ADF introduz uma etapa de agregação dos modelos dos clientes. Essa etapa geralmente é realizada por um servidor de agregação, então também pode ser otimizada para tal.

Cada cliente é responsável por armazenar e realizar o treino apenas do modelo do cliente. Logo, o custo de memória no cliente continua sendo o mesmo em relação ao AD. Já o servidor de agregação precisa de memória suficiente para armazenar e aplicar a função de agregação em todos os modelos dos clientes que participam do treino em dada iteração.

## 2.6 Quantização

Os avanços obtidos por soluções de aprendizado profundo têm-se intensificado. No entanto, houve também um crescimento na sua complexidade, incluindo o maior número de parâmetros e arquiteturas mais complexas, exigindo um alto poder de processamento para sua aplicação (Lê *et al.*, 2023). Esse cenário trouxe a necessidade do desenvolvimento de técnicas que tornem a aplicação de redes neurais mais eficiente em termos de recursos computacionais

como processamento, consumo de memória, rede, energia, dentre outros.

Dentre essas técnicas, se destacam atualmente a quantização, destilação e poda (*pruning*) (Lê *et al.*, 2023).O presente trabalho aborda a quantização. No contexto de aprendizado de máquina, quantização é uma técnica que consiste em realizar operações aritméticas utilizando representações numéricas de baixa precisão com o propósito de reduzir o consumo de memória, processamento e energia da máquina (NAGEL *et al.*, 2021). Por exemplo, ao representar os parâmetros de uma rede neural como valores inteiros de 8 bits, ao invés da representação tradicional com ponto flutuante de 32 bits, a taxa de transferência de memória para o processador e o tamanho do modelo podem ser reduzidos em até 4 vezes (Lê *et al.*, 2023).

A quantização também pode beneficiar o tempo de execução de operações como multiplicação de matrizes, convoluções e soma de vetores. Operações realizadas com inteiros de 8 bits tendem a ser mais otimizadas a nível de *hardware* do que operações com valores ponto flutuante de 32 bits (GHOLAMI *et al.*, 2022). Por consequência, ocorre também a redução no consumo de energia necessário para a execução do algoritmo.

Por um outro lado, essa redução na precisão numérica dos parâmetros do modelo pode introduzir erros de arredondamento e prejudicar a qualidade da inferência do modelo. Um dos principais desafios da área da quantização é minimizar o impacto negativo à acurácia do modelo (GHOLAMI *et al.*, 2022). Para isso, foram desenvolvidas técnicas de quantização que geralmente envolvem estatística e agrupamento de parâmetros para encontrar uma representação no espaço limitado de quantização que minimize o erro.

### *2.6.1 Representação com precisão completa*

A representação de ponto flutuante mais utilizada é denominada representação polarizada (STALLINGS, 2009). Nessa representação, um número real X é representado através de três palavras binárias S, M e E que juntas formam uma palavra binária *SME* definida na Equação 2.3.

$$X = (-1)^S * M * 2^E \tag{2.3}$$

Onde:

- $S$ é o bit de sinal, sendo 0 para números positivos e 1 para os negativos.

- $M$ é o denominada a mantissa, uma palavra binária responsável por armazenar o significado do número.
- $E$ é o expoente inteiro na base 2, representando a amplitude do número.

O tamanho de uma palavra binária utilizada para representar um número real depende de quantos bits são utilizados para representar a mantissa e o expoente. Quanto maior a quantidade de bits, maior será a faixa e a precisão dos números representados e, de forma equivalente, maior será o custo computacional envolvido para se armazenar os seus valores e realizar as operações aritméticas (STALLINGS, 2009). As representações de ponto flutuante mais utilizadas atualmente são definidas pela *IEE Standard 754* (IEEE. . . , 2019). Esse padrão define um ponto flutuante de 32 bits como uma composição de 1 bit de sinal, 8 bits para o expoente e 23 bits para a mantissa.

### *2.6.2 Representação quantizada*

Estudos na área exploram a quantização numérica em diferentes níveis de precisão, incluindo 8 bits, 4 bits, 2 bits e 1 bit, sendo a representação em inteiros de 8 bits a mais comum. Essa representação busca equilibrar a redução de custo computacional e preservação da acurácia do modelo (WU *et al.*, 2020; KRISHNAMOORTHI, 2018).

Porém, a redução da representação numérica de 32 bits para 8 bits é um desafio que vai muito além de simplesmente diminuir o tamanho da mantissa e do expoente. Um número ponto flutuante de 32 bits pode representar em torno de 4,3 bilhões de valores distintos, enquanto um número de 8 bits pode representar apenas 256 números distintos. Essa redução drástica faz com que a quantização tenha que ser projetada com cautela para aproveitar ao máximo cada um desses 256 valores distintos. O objetivo da quantização é minimizar o erro e ao mesmo tempo manter eficiência nas operações aritméticas (WENG, 2023).

### *2.6.3 Quantização uniforme*

Uma das técnicas mais comuns para conversão de valores reais para valores quantizados consiste na quantização uniforme (também chamada de quantização uniforme assimétrica), onde cada valor real é mapeado para um valor inteiro através da aplicação de uma transformação linear ou uma transformação afim, seguida de uma operação de arredondamento (KRISHNAMOORTHI, 2018).

Na quantização em geral, um conjunto de valores reais dentro de um intervalo

$(r_{min}, r_{max})$ precisam ser representados como números inteiros dentro de um intervalo $(0, 2^b - 1)$, onde $b$ é o número de bits utilizados na representação quantizada. Na quantização uniforme afim, é definido uma função de quantização $Q(r)$ que mapeia um valor real $r$ para o intervalo quantizado (Equações 2.4 e 2.5).

$$r_{\text{int}} = \text{ARREDONDAR}\left(\frac{r}{S}\right) + Z \tag{2.4}$$

$$Q(r) = \text{LIMITAR}(r_{int}, 0, 2^b - 1) \tag{2.5}$$

Onde:

- $S$ é o fator de escala, responsável por ajustar a largura da faixa de números reais para o espaço quantizado. Pode ser calculado como $S = (x_{max} - x_{min})/(2^b - 1)$.
- $Z$ é um valor inteiro que desloca o intervalo quantizado. É utilizado para garantir que o menor número representado corresponda ao valor 0 no espaço quantizado. Pode ser calculado como $Z = -arredondar(r_{min}/S)$.
- $ARREDONDAR(r)$ é uma função que converte um número real em um número inteiro próximo. A função de teto é um exemplo desse tipo de função.
- $LIMITAR(r_{int}, 0, 2^b - 1)$ é uma função que restringe o valor $r_{int}$ ao intervalo válido entre 0 e $2^b - 1$. Se $r_{int}$ for maior que o máximo permitido $(2^b - 1)$, ele é ajustado para $2^b - 1$. Se for menor que 0, é ajustado para 0.

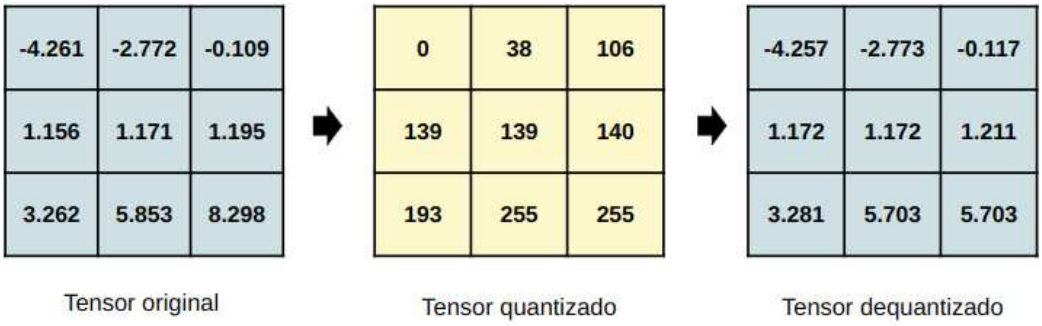
Note que o valor original de um número pode ser perdido ao aplicar as funções de *arredondar* e *limitar*, tornando a função $Q(r)$ irreversível. No entanto, ainda é possível reconstruir uma aproximação do valor original a partir do valor inteiro quantizado $r_q$ através da função de dequantização (Equação 2.6).

$$DQ(r_q) = (r_q - Z) * S \tag{2.6}$$

A Figura 7 ilustra um exemplo de uma matriz quantizada e dequantizada logo em seguida. A diferença entre o valor original $r$ em ponto flutuante e o valor reconstruído $DQ(Q(r))$ após a quantização e dequantização é denominada erro de quantização.

A escolha adequada dos parâmetros de quantização (nesse caso, os valores de $S$ e $Z$) é importante para reduzir esse erro de quantização. Nas redes neurais, esses parâmetros

Figura 7 – Exemplo de um tensor inicialmente composto de números reais sendo quantizado e dequantizado. Os parâmetros de quantização utilizados foram de $S = 0.039$ e $Z = 109$. Note que o tensor dequantizado não é igual ao original devido ao erro de quantização.




Fonte: Elaborado pelo autor.

devem ser calculados levando em consideração a distribuição dos pesos e ativações gerados pela propagação dos dados de entrada. Para isso, é realizada uma etapa denominada *calibração*, onde o modelo é observado enquanto realiza a operação de propagação para um subconjunto de dados específico denominado conjunto de dados de calibração. Durante essa propagação, dados estatísticos como a média, o desvio padrão, os quartis dos pesos e ativações são monitorados. Posteriormente, essas informações são utilizadas para calcular os parâmetros da quantização.

Existem outras técnicas de quantização além da quantização uniforme afim. Outro exemplo é a quantização uniforme simétrica, onde o valor de $Z$ é fixo em zero e o intervalo é centrado em zero, ou seja, o valor de $x_{min} = -x_{max}$. Dessa forma, a operação de somar $Z$ é eliminada, mas o erro de quantização é drasticamente aumentado em distribuições que não são simétricas em torno de zero. Outra técnica é a quantização não uniforme, onde funções mais complexas são utilizadas para mapear o valor real para o inteiro. Exemplos incluem funções quadráticas, logarítmicas e funções de clusterização (WENG, 2023). Apesar de proporcionarem maior flexibilidade, essas técnicas também causam um maior custo computacional.

### *2.6.4 Granularidade*

Na quantização uniforme, cada peso ou ativação quantizado de uma rede neural precisa estar associado a seus parâmetros de quantização $S$ e $Z$. Porém, se para cada valor que será quantizado for criado um par de parâmetros $S$ e $Z$, a sobrecarga causada por manter esses parâmetros remove todos os benefícios que a quantização poderia trazer. Por outro lado, compartilhar apenas um valor de $S$ e $Z$ entre todos os parâmetros do modelo pode trazer um alto

erro de quantização. Isso ocorre por que pesos e ativações dentro de uma mesma rede neural podem ter distribuições estatísticas muito distintas (WENG, 2023).

Para evitar esse problema, os pesos e ativações da rede neural são quantizados em grupos, onde cada grupo compartilha os mesmos valores de $S$ e $Z$. Dessa forma, cada grupo tem seus parâmetros de quantização mais bem ajustados à distribuição de seus valores. Uma forma comum de agrupar os parâmetros em redes neurais é unir valores dentro de cada camada do modelo. Alternativamente, camadas convolucionais podem agrupar os valores por canais dentro da camada (KRISHNAMOORTHI, 2018).

### *2.6.5 Quantização estática e dinâmica*

O cálculo dos parâmetros de quantização pode ser realizado de forma estática ou dinâmica (GHOLAMI *et al.*, 2022). A quantização estática consiste em pré-definir os parâmetros de quantização antes da etapa de inferência. Já a quantização dinâmica define os parâmetros de quantização em tempo de inferência, ou seja, os pesos ou ativações são gerados com precisão completa e, posteriormente, são de fato quantizados.

A escolha entre quantização estática e quantização dinâmica depende de fatores como a arquitetura da rede neural, a distribuição dos dados de entrada e o ambiente computacional (GHOLAMI *et al.*, 2022). A quantização estática tende a causar um maior erro de quantização, pois os parâmetros de quantização são estimados a partir de um conjunto de calibração e aplicados a dados ainda não vistos. Porém, ela reduz os custos computacionais ao permitir reutilizar os parâmetros quantizados durante a inferência.

### *2.6.6 Ajuste dos parâmetros de quantização*

A quantização estática requer a estimativa dos parâmetros de quantização antes de o modelo ser propriamente quantizado. Esse cálculo é denominado como calibração, onde os parâmetros de quantização são calculados através da propagação de dados nas redes neurais (NAGEL *et al.*, 2021). Valores estatísticos como média, desvio padrão, mínimo, máximo e quartis dos pesos e ativações são observados durante a inferência de um conjunto de dados denominado conjunto de dados de calibração. Posteriormente, essas informações são utilizadas para o cálculo dos parâmetros de quantização. Há duas abordagens predominantes para calibração: Quantização Pós Treino (QPT) e Treino Consciente de Quantização (TCQ).

#### *2.6.6.1 Quantização pós treino*

Na QPT, os parâmetros de quantização são calculados após o modelo ter sido treinado com precisão completa. Sendo necessário apenas uma rodada extra de propagação do conjunto de dados de calibração para realizar a calibração (NAGEL *et al.*, 2021).

Essa abordagem requer menos esforço e recursos em relação ao TCQ, pois a propagação dos dados do conjunto de calibração é suficiente para calibração. No entanto, é mais suscetível a perdas de acurácia do modelo (NAGEL *et al.*, 2021).

#### *2.6.6.2 Treinamento consciente de quantização*

No TCQ, a calibração interfere no processo de treino da rede neural. A técnica consiste em simular pesos e ativações quantizados na propagação dos dados para que o gradiente calculado na retropropagação do erro se adapte à versão quantizada do modelo (NAGEL *et al.*, 2021). Dessa forma, a calibração é realizada durante o treino e, ao mesmo tempo, o treino se adapta aos pesos e ativações quantizados.

O TCQ proporciona uma acurácia mais próxima à do modelo original. No entanto, essa abordagem exige um maior custo computacional, pois requer uma etapa adicional de treinamento com quantização simulada (NAGEL *et al.*, 2021).

### *2.6.7 Treino com baixa precisão*

É importante destacar que o trabalho aqui proposto aborda a quantização somente para inferência, ou seja, a redução dos custos computacionais não afeta a etapa de treino da rede neural.

Atualmente, maior parte do esforço destinado à quantização se concentra na otimização da inferência, sem abordar a quantização do treino em si (WENG, 2023; KULKARNI *et al.*, 2022). Para o treino quantizado (ou treino com baixa precisão), é necessário quantizar os pesos, as ativações e os gradientes da retropropagação e, apesar dos avanços na área, a quantização no treino para representações abaixo de 16 bits tem-se mostrado um grande desafio devido à instabilidade numérica.

## 2.7 Trabalhos relacionados

Até o momento da escrita deste trabalho, não foram identificados outros trabalhos que apresentem uma avaliação de desempenho especificamente da quantização do modelo do cliente no contexto de aprendizado dividido federado, o que indica uma lacuna na literatura atual. O trabalho proposto é o primeiro a empregar diferentes configurações de modelos, conjuntos de dados, camadas de corte e esquemas de quantização para analisar o impacto no cliente.

As seguintes subseções detalham os trabalhos relacionados mais próximos a presente pesquisa. Também são resumidos os demais estudos que aplicam técnicas de quantização de diferentes formas para a redução de custos computacionais em cenários de AD ou ADF.

### *2.7.1 PEACE: Private and Energy-Efficient Algorithm for Cardiac Evaluation on the EDGE using Modified Split Learning and Model Quantization*

O trabalho de Ayad *et al.* (2023) utiliza a quantização de forma semelhante à proposta desta dissertação: o modelo do cliente é quantizado para reduzir custos computacionais. Porém, o trabalho foca em uma solução de AD específica e aplica uma série de modificações para tornar o sistema mais eficiente na classificação de dados de eletrocardiogramas. Uma dessas modificações é justamente a quantização do modelo do cliente para reduzir seus custos de inferência, como proposto nesta dissertação.

Em resumo, Ayad *et al.* (2023) propõe três técnicas para a otimização do AD:

- Adição de um *autoencoder* para codificar a saída do modelo do cliente e de um *decoder* no lado do servidor para decodificação. Isso possibilita a redução de dimensionalidade dos dados enviados pelo cliente ao servidor, reduzindo também a comunicação entre eles.
- Durante o treino, a atualização do modelo do cliente ocorre somente quando o valor dos gradientes ultrapassa um determinado nível de tolerância, reduzindo os custos de retropropagação do modelo do cliente para mudanças pequenas.
- Quantização do modelo para o cliente após o treino, com diferentes configurações de 16, 8 e 4 bits.

Após aplicar essas técnicas de *autoencoder* e a tolerância à passagem do gradiente, observou-se que o modelo com precisão completa de 32 bits apresentou redução de 1,5% na acurácia em comparação ao algoritmo de AD tradicional. Observou-se que os modelos com 16, 8 e 4 bits sofreram, respectivamente, perdas adicionais de acurácia de 0%, 1% e 22%.

Até o momento da escrita deste trabalho, esse foi o único trabalho encontrado que utiliza a técnica de quantização de 8 bits para aprimorar a inferência do modelo do cliente no contexto de AD. Embora a técnica seja explorada neste estudo, ainda há pouca evidência que comprove sua eficácia e generalização em diferentes configurações de modelos, uma vez que a solução foi testada apenas para um problema específico de classificação de séries temporais e utilizando uma arquitetura de rede neural específica para classificação de eletrocardiogramas.

O presente trabalho busca expandir o estudo da quantização ao analisá-la individualmente em diferentes combinações de conjuntos de dados e de modelos de visão computacional no cenário de ADF. Além disso, serão explorados diferentes esquemas de quantização e diferentes métricas de desempenho.

### *2.7.2 Binarizing Split Learning for Data Privacy Enhancement and Computation Reduction*

O trabalho de Pham *et al.* (2023) propõe o uso de redes neurais binárias no lado do cliente para melhorar a eficiência e a privacidade do aprendizado dividido. Esse método, denominado *B-SL*, consiste em utilizar pesos e ativações binários, em que os valores são restritos a -1 ou +1. Essa técnica reduz o tempo de inferência e de treino, o consumo de memória e a comunicação, sem degradar significativamente a acurácia.

Além disso, a técnica busca reduzir o potencial de vazamento de informações nos dados enviados pelo cliente ao servidor (mapa de características), incorporando duas técnicas adicionais: aplicar, no treino, uma penalização pelo vazamento de informações e adotar uma técnica de privacidade diferencial nos dados enviados ao servidor.

Suas técnicas são validadas por meio de experimentos com problemas de classificação de imagens em quatro conjuntos de dados: *MNIST*, *Fashion-MNIST*, *SVHN* e *CIFAR-10*. As redes neurais utilizadas são o *VGG* (modificado para aceitar resoluções de 32x32) com o *CIFAR-10* e a *LeNet-5* para os demais conjuntos de dados. Os resultados indicam uma redução de tempo de propagação de até 17,5 vezes e redução no consumo máximo de memória e largura de banda de até um máximo de 32 vezes.

Apesar do resultados positivos, o trabalho de Ayad *et al.* (2023) possui limitações. Por exemplo, ao dividir o modelo entre o cliente e o servidor, apenas a camada de entrada e a primeira camada convolucional foram alocadas ao cliente nos experimentos de eficiência computacional. Falta então demonstrar a eficácia da binarização em diferentes configurações de modelos e de camadas de corte.

Tabela 1 – Configurações dos experimentos de Dachille *et al.* (2024).

| Conjunto de dados | Modelo | Número de clientes | Rodadas |
|---|---|---|---|
| HAM10000 | ResNet-18 | 100 | 100 |
| CIFAR-10 | ResNet-18 | 100 | 200 |
| CIFAR-100 | ResNet-50 | 100 | 300 |
| Tiny ImageNet | ResNet-50 | 10 | 300 |

Fonte: Adaptado de Dachille *et al.* (2024).

Além disso, o trabalho apenas assume que a quantização é teoricamente capaz de reduzir o consumo de memória e de rede em até 32 vezes, sem realizar medições nos experimentos, ignorando possíveis sobrecargas da quantização e demais sobrecargas da solução da rede neural.

### *2.7.3 The Impact of Cut Layer Selection in Split Federated Learning*

O trabalho de Dachille *et al.* (2024) realiza uma análise empírica e teórica sobre como a seleção da camada de corte influencia o desempenho e a convergência no ADF. São apresentadas duas variantes do ADF: SFL-V1 e SFL-V2. No SFL-V1, o servidor de mantém um modelo de servidor distinto para cada cliente, que são agregados ao final de cada rodada de comunicação. Já no SFL-V2 (o mais comum e foco do presente trabalho), o servidor utiliza um único modelo compartilhado que processa simultaneamente as ativações de todos os clientes.

De forma semelhante ao presente trabalho, Dachille *et al.* (2024) busca entregar uma análise empírica de desempenho computacional em um cenário de ADF. No entanto trabalho não aborda quantização o uso de quantização, ao invés disso, limitando-se a variar a posição da camada de corte para observar o impacto na acurácia da solução. A Tabela 1 exibe as diferentes configurações de conjuntos de dados, modelos, número de clientes e rodadas de treino abordados no trabalho. Além disso, versões IID e não-IID foram geradas dos conjunto de dados.

Enquanto Dachille *et al.* (2024) foca na acurácia e na análise de convergência teórica, esta dissertação realiza a coleta de diversas métricas além da precisão do modelo. Em nosso trabalho, é avaliado o tempo de propagação do cliente, o pico de consumo de memória, o consumo de rede, o tamanho do modelo e a acurácia global. Além disso, nossos experimentos utilizam um ambiente distribuído real configurado via Google Remote Procedure Call (gRPC) em máquinas virtuais na nuvem para mimetizar as restrições de infraestrutura.

### *2.7.4 Demais trabalhos relacionados*

A Tabela 2 resume os demais trabalhos relacionados encontrados durante a concepção deste estudo. Estes, por suas vez, analisam o AD e o ADF com uma aplicação diferente da quantização do modelo do cliente.

Tabela 2 – Trabalhos relacionados.

| Autor | Título | Propósito do estudo | Cenário |
|---|---|---|---|
| (JOSHI *et al.*, 2023) | Enabling All In-Edge Deep Learning: A Literature Review | Realiza uma revisão da literatura sobre o treino e inferência de modelos de aprendizado profundo em sistemas com baixo poder computacional. Dentre as técnicas citadas, estão o aprendizado dividido e quantização. | AD |
| (DUAN *et al.*, 2023) | Efficient Federated Learning Method for Cloud-Edge Network Communication | Agrupamento de clientes pela similaridade da distribuição de dados para reduzir os custos de se treinar muitos clientes em paralelo. Além disso, o estudo aplica quantização do IR para reduzir ainda mais custos de comunicação. | ADF |
| (CHEN *et al.*, 2021) | Communication and Computation Reduction for Split Learning using Asynchronous Training | O estudo aplica uma técnica para filtrar atualizações nos clientes: o cliente é atualizado apenas quando o gradiente do IR retornado pelo servidor é maior que um certo limite. Além disso, é aplicado a quantização de 8 bits do IR e gradiente do IR para redução de comunicação. | AD |
| (GUPTA *et al.*, 2026) | Adaptive Low-Latency Split Federated Learning With Dynamic Model Partitioning in Resource-Constrained Healthcare IoT | Propõe o uso de Aprendizado Dividido Federado Dinâmico que realiza a escolha da camada de corte de acordo com o contexto do dispositivo do cliente. Nesse estudo, também é aplicada a quantização para redução da comunicação. | ADF |
| (ZHANG *et al.*, 2025a) | Federated Split Learning With Model Pruning and Gradient Quantization in Wireless Networks | O trabalho propõe a poda do modelo e quantização dos gradientes para redução de custos computacionais no cenário de ADF. Além disso, é aplicado um dropout aleatório para redução do IR. | ADF |
| (OH *et al.*, 2025) | Communication-Efficient Split Learning via Adaptive Feature-Wise Compression | O trabalho propõe duas estratégias para redução de comunicação entre o cliente o servidor e os clientes. A primeira consiste em descartar probabilisticamente características baseando-se em seu desvio padrão. A segunda consiste em quantizar os vetores de características restantes de acordo com sua variação. | AD |
| (ZHANG *et al.*, 2025b) | Split Fine-Tuning for Large Language Models in Wireless Networks | O trabalho propõe um esquema de ajuste fino (*fine-tuning*) de modelos de linguagem grandes (LLMs) em dispositivos móveis com recursos limitados. Dentre as técnicas abordadas, a quantização do IR é aplicada para redução de comunicação entre cliente e servidor. | AD |

Fonte: Elaborado pelo autor.

# 3 PROCEDIMENTOS METODOLÓGICOS

O presente trabalho avalia o impacto da quantização do modelo do cliente na redução dos custos computacionais em soluções de Aprendizado Dividido Federado (ADF). Para validar o uso dessa técnica, uma série de experimentos será realizada com diferentes configurações em um ambiente que simula um cenário de aprendizado colaborativo. Os experimentos terão foco em tarefas de visão computacional, a tarefa mais frequente em aprendizado colaborativo.

Este capítulo aborda como serão executados os experimentos e a análise de seus resultados. As próximas seções descrevem detalhes sobre os conjuntos de dados, arquiteturas de redes neurais, técnicas de quantização, ambiente computacional e, por fim, as métricas que serão coletadas. O código fonte necessário para replicar os experimentos está documentado e disponível em um repositório público do GitHub [1].

## 3.1 Conjunto de Dados

Para o treino das redes neurais, é utilizado o conjunto de dados *CIFAR10* em duas versões IID e não-IID. Desenvolvido pela *Canadian Institute for Advanced Research (CIFAR)*, o *CIFAR10* é um conjunto de dados composto por 60.000 imagens coloridas, distribuídas em 10 classes diversificadas (KRIZHEVSKY; HINTON, 2009). O conjunto é subdividido em dois subconjuntos: conjunto de treino (50.000 imagens) e conjunto de teste (10.000 imagens) igualmente distribuídos entre as 10 classes. A resolução original dessas imagens é de 32x32, mas para este trabalho, elas foram redimensionadas para 224x224 para estarem compatíveis com as redes neurais. A Figura 8 ilustra algumas das instâncias distribuídas nessas 10 classes.

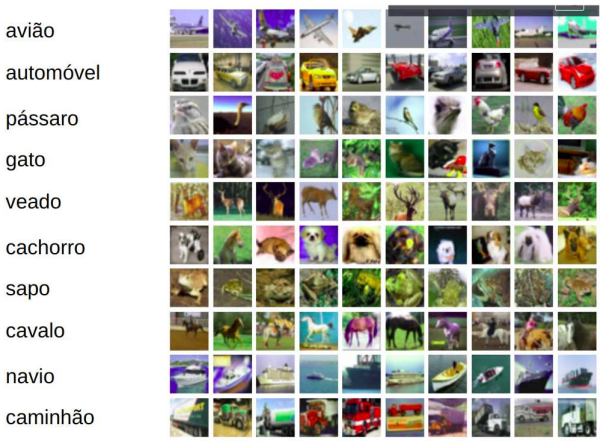
Esse conjunto de dados foi escolhido devido a sua popularidade que facilita a comparação com demais trabalhos presentes na literatura.

## 3.2 Distribuição dos dados

Em um cenário comum de aprendizado colaborativo, dados obtidos de diferentes tendem a ser distintos entre si, gerando uma heterogeneidade nos dados que desafia o treino de soluções colaborativas (KAIROUZ *et al.*, 2021). Ao mesmo tempo, a literatura sofre de uma falta de conjuntos de dados públicos que representem cenários descentralizados reais (G. *et al.*, 2024). Uma forma comum de contornar isso é particionar um conjunto de dados centralizado de

[1] https://github.com/ClaroHenrique/QFSL-benchmark

Figura 8 – Exemplo de imagens separadas por classe do conjunto de dados CIFAR-10.




Fonte: Adaptado de Krizhevsky e Hinton (2009)

uma forma que simule uma distribuição não-IID. Nos experimentos realizados, foi gerado uma versão IID e uma não-IID do CIFAR-10.

No conjunto de dados CIFAR-10 IID, cada instância foi atribuída a um cliente sorteado com probabilidade uniforme entre os clientes. Na versão não-IID, as classes foram distribuídas de forma desbalanceada. Para isso, cada classe recebe uma distribuição (que foi gerada por uma Dirichlet com $\alpha = 0,1$) que indica a probabilidade de uma instância dessa classe ser alocada para determinado cliente.

Os conjuntos de dados também foram gerados considerando os casos com 4 e 8 clientes. Portanto, 4 combinações foram geradas no total, alternando o número de clientes $N = \{4, 8\}$ e distribuições IID e não-IID. A Figura 9 ilustra a diferença entre as distribuições dos dados para 4 clientes e 8 clientes. Note que o caso com 8 clientes é mais desbalanceado.

## 3.3 Modelos

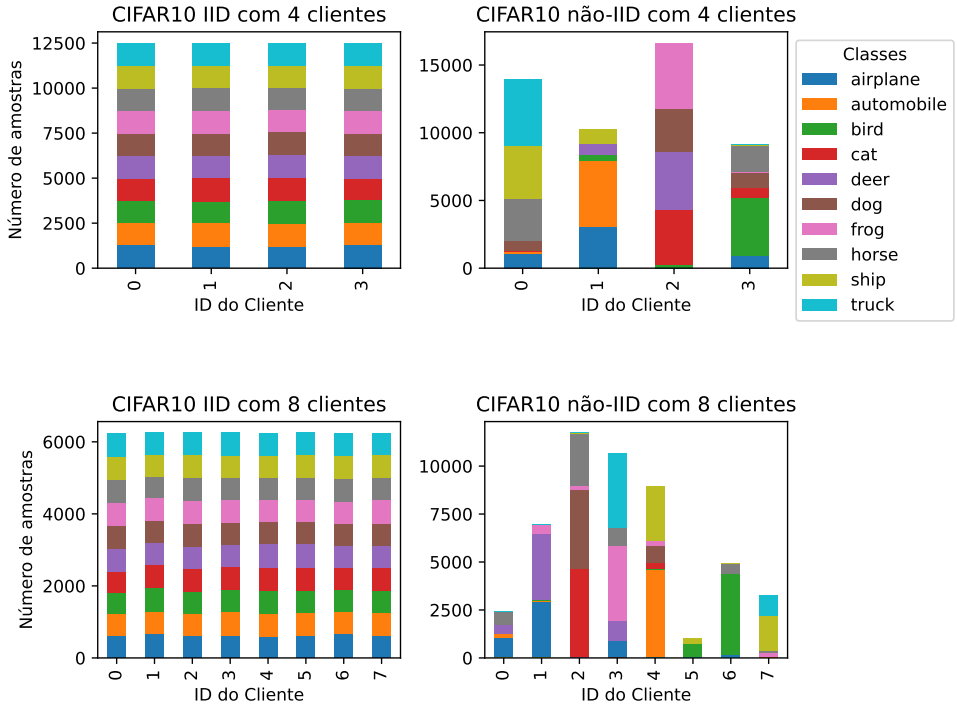
Para esse trabalho, foram escolhidos modelos de RNCs por serem atualmente a solução no estado da arte para problemas de classificação de imagens.

### *3.3.1 VGG11*

A família de redes Visual Geometry Group (VGG) são arquiteturas de redes neurais convolucionais introduzidas por Simonyan e Zisserman (2015) e criadas com o propósito de aumentar a profundidade da rede sem aumentar drasticamente o número de parâmetros.

A arquitetura consiste no uso de filtros de convolução (3x3) e operações de *pooling*

Figura 9 – Comparação entre a distribuição do conjunto CIFAR10 IID e CIFAR10 não-IID para 4 e 8 clientes




Fonte: Elaborado pelo autor.

aplicadas sequencialmente até chegar nas últimas camadas, quando a imagem intermediária é convertida para uma representação linear e então processada por camadas densas para classificação (SIMONYAN; ZISSERMAN, 2015). No presente trabalho será utilizada a rede VGG11 que possui no total 11 camadas, composta por 8 camadas convolucionais e 3 camadas densas, seguidas de funções de ativação ReLU e camadas de *pooling* para reduzir a dimensionalidade.

### *3.3.2 ResNet18*

A família de rede neurais ResNet (*Residual Network*) foi introduzida por He *et al.* (2016) e projetada para mitigar o problema de dissipação do gradiente que ocorre na medida em que a profundidade da rede neural aumenta.

Para isso, a rede introduz o conceito de conexões residuais (ou conexões de atalho) que permitem que a entrada original da rede seja propagada diretamente para camadas mais profundas. A rede é então particionada em blocos residuais, onde cada bloco possui uma conexão residual entre sua primeira e última camada He *et al.* (2016).

A arquitetura abordada neste trabalho é a *ResNet18* composta por 18 camadas, sendo a primeira camada uma convolução, seguida de 4 blocos residuais com 4 camadas convolucionais

cada e uma última camada densa para classificação. Embora a *ResNet18* tenha um número maior de camadas em relação à *VGG11*, ela possui convoluções menores e mais otimizadas e mantém a acurácia mesmo sendo bem mais leve.

### *3.3.3 MobileNetV2*

A arquitetura de rede neural *MobileNet* foi introduzida por Howard *et al.* (2017) com o propósito de otimizar redes neurais para ambientes com recursos de *hardware* limitados, como, por exemplo, dispositivos móveis e IoT.

A família *MobileNet* foi projetada para ser leve e eficiente através do uso convoluções separáveis em profundidade, combinado de convoluções em profundidade e convoluções ponto a ponto. Essa abordagem reduz de forma considerável o número de parâmetros e custos computacionais da rede.

O presente trabalho utiliza especificamente a arquitetura *MobileNetV2*. Em comparação com as demais redes, a inovação da *MobileNetV2* está no foco em sua eficiência computacional, buscando manter uma boa acurácia mesmo com um número reduzido de parâmetros.

### *3.3.4 Divisão dos modelos*

Para cada uma dos modelos apresentados, dois pontos da camada de corte foram selecionados para particionar a rede entre cliente e servidor. Em todos os casos, o cliente armazena as camadas convolucionais iniciais, incluindo a camada de entrada. Essa divisão é feita de acordo com um parâmetro $S \in \{1,2\}$, onde $S = 1$ representa um ponto de corte mais raso e $S = 2$ representa um ponto de corte com mais camadas no cliente. Essa divisão é detalhada na Tabela 3, enquanto a Figura 10 exemplifica a divisão do modelo *ResNet18*.

Tabela 3 – Seleção da camada de corte das arquiteturas utilizadas.

| Modelo | Camada de corte $S = 1$ | Camada de corte $S = 2$ |
|---|---|---|
| VGG11 | Até a primeira camada do modelo | Até a segunda camada do modelo |
| ResNet18 | Até o primeiro bloco residual (5 camadas) | Até o segundo bloco residual (9 camadas) |
| MobileNetV2 | Até o primeiro bloco residual inverso (2 camadas) | Até o segundo bloco residual inverso (8 camadas) |

Fonte: elaborado pelo autor.

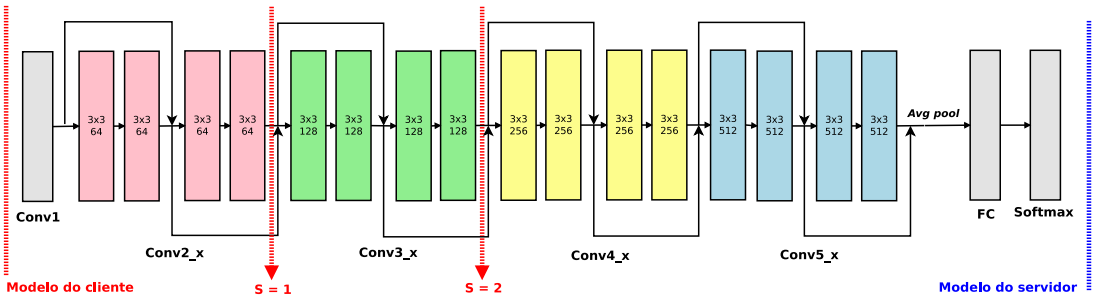



Figura 10 – A divisão entre modelo do cliente e modelo do servidor da arquitetura *ResNet18* foi realizada com o primeiro bloco residual ($S = 1$) e segundo bloco residual ($S = 2$).

## 3.4 Biblioteca de aprendizado de máquina

O presente trabalho implementa os modelos de redes neurais e quantização através do *framework Pytorch*. O *Pytorch*, desenvolvido pelo *Facebook's AI Research lab (FAIR)*, é a implementação em *Python* do *Torch*, uma biblioteca inicialmente desenvolvida em C que disponibiliza um ambiente para manipulação de matrizes multidimensionais para Machine Learning (ML) dentre outras aplicações matemáticas (YEGULALP, 2017).

O *Pytorch* permite que os desenvolvedores definam manualmente o valor do gradiente utilizado na etapa de retropropagação no treino dos modelos (ANSEL *et al.*, 2024a). Esse nível de controle sobre o algoritmo possibilita a implementação do ADF sem a necessidade de modificações sobre o *framework*.

## 3.5 Treino dos modelos e quantização

Em relação à quantização, foram aplicadas as configurações de QPT e TCQ ao modelo do cliente com o objetivo de reduzir seus custos de inferência. Para isso, foi utilizada a implementação padrão da biblioteca *Pytorch*, com suporte a ambas as técnicas (ANSEL *et al.*, 2024b).

O custo de memória de redes neurais convolucionais é dominado pelas ativações (mapas de características) geradas durante a inferência, enquanto o custo de tempo é dominado pela convolução entre os pesos e as ativações (KRISHNAMOORTHI, 2018). Logo, como o modelo do cliente neste trabalho é composto apenas por camadas convolucionais, percebeu-se a necessidade da quantização estática. Na quantização estática, os pesos e as ativações permanecem com baixa precisão durante toda a inferência.

No presente trabalho, três versões de cada modelo são gerados para comparação: um com precisão completa, um modelo com QPT e um modelo com TCQ. O modelo com precisão

completa foi obtido após um treino em 200 épocas com 32 bits. O modelo com QPT foi gerado através da quantização direta do modelo com precisão completa, sem a necessidade de treino extra.

Enquanto o modelo com TCQ foi obtido através do treino modelo com precisão completa de 180 épocas, seguido de um ajuste fino de 20 épocas para o modelo para a quantização.

## 3.6 Ambiente computacional

As redes neurais foram treinadas em uma uma máquina robusta com acesso a *GPU* disponibilizada em nosso laboratório de pesquisa no DC/UFC. Esse treino foi realizado simulando um cenário de ADF com a distribuição dos dados entre clientes, divisão do modelo entre cliente e servidor.

Para nos experimentos de inferência, o servidor permaneceu o mesmo, enquanto os clientes foram alocados em máquinas virtuais instanciadas sobre a infraestrutura Infrastructure as a Service (IaaS) oferecida pelo serviço da *Google Cloud*. A plataforma permite a implantação de aplicações através do aluguel de máquinas virtuais, além de oferecer controle sobre os recursos computacionais de cada máquina.

A comunicação entre a máquina do servidor e a máquina do cliente foi implementada através do protocolo *gRPC*, um protocolo de comunicação eficiente e robusto, que permite a uma máquina executar funções ou métodos em outra máquina de forma transparente, simulando funções locais.

## 3.7 Métricas

Para medir os efeitos da quantização do modelo do cliente, foram coletadas as seguintes métricas de desempenho computacional: o tempo de execução, o consumo de memória, o tamanho do modelo e o consumo de rede. Como o foco deste trabalho é otimizar a inferência no lado do cliente, essas métricas serão coletadas no cliente. Além disso, foi analisado também o impacto causado pela redução da precisão na acurácia do modelo total.

### *3.7.1 Tempo de propagação do cliente*

O impacto da quantização na velocidade da inferência no lado do cliente será avaliado medindo o tempo de execução desde o início da etapa de propagação na camada de entrada até a

camada de corte, onde será gerado o IR que será enviado ao servidor.

### *3.7.2 Pico de memória do cliente*

O pico de memória foi avaliado pelo uso máximo de memória medido durante a propagação na máquina do cliente. Ferramentas nativas de medição de memória da biblioteca *Pytorch* não oferecem suporte aos modelos quantizados executados na CPU. Ao invés disso, foi utilizada a biblioteca *Memory Profiler*, que coleta o uso de memória do processo (e seus subprocessos) a nível de sistema operacional.

### *3.7.3 Tamanho de modelo do cliente*

O tamanho do modelo do cliente será medido em bytes, sendo o espaço necessário para armazená-lo na máquina cliente, incluindo seus parâmetros e metadados. Reduzir o tamanho do modelo é conveniente para dispositivos com baixa capacidade de armazenamento, como em dispositivos móveis e sistemas IoT. Além disso, a redução do tamanho do modelo pode mitigar o custo de comunicação em modelos transferidos durante operações de agregação do ADF.

### *3.7.4 Consumo de rede*

O consumo de rede será avaliado pela quantidade total de dados (em megabytes) transferidos entre o cliente e o servidor durante a inferência. Essa métrica é importante em ambientes com largura de banda limitada, como redes móveis.

### *3.7.5 Acurácia*

Com a redução na representação numérica dos pesos e ativações do modelo quantizado no cliente, é esperado também uma perda na acurácia do modelo global. Medir a acurácia do modelo antes e depois da quantização sobre o conjunto de dados de teste nos permite analisar o custo-benefício da quantização.

### *3.7.6 Baseline*

Para avaliar as vantagens e desvantagens da quantização, utilizamos como *baseline* a versão do modelo com precisão completa. Assim, o desempenho das soluções quantizadas de 8 bits geradas através do QPT e TCQ foi comparado com a solução com precisão completa de 32

bits, usando as métricas citadas anteriormente.

## 3.8 Categorias dos experimentos

Os experimentos foram estruturados em duas categorias: análise de acurácia e análise de eficiência computacional. Os testes voltados à eficiência computacional foram submetidos a 30 repetições enquanto, por limitações de tempo, os testes de acurácia foram repetidos 10 vezes.

Na categoria de acurácia, os modelos são treinados considerando combinações diferentes conjuntos de dados, número de clientes e esquemas de quantização. Porém, a escolha dessas configurações influencia apenas nos valores numéricos dos parâmetros do modelo calculados durante o treino.

As métricas de tempo de execução, consumo de memória, consumo de rede e tamanho do modelo não são influenciadas pelos valores numéricos dos parâmetros dos modelos ou dados de entrada, pois a computação é a mesma. Isso motivou a separação dos experimentos de eficiência computacional, onde foi fixado o modelo gerado após o treino no conjunto de dados CIFAR-10 IID, número de clientes igual a 4 e com o esquema QPT.

# 4 RESULTADOS E DISCUSSÕES

O presente capítulo tem como propósito apresentar e discutir os resultados medidos após a execução dos experimentos. A Seção 4.1 foca em avaliar os ganhos de eficiência de custo computacional, explorando as demais métricas como tempo de execução e memória, enquanto a Seção 4.2 descreve como a quantização do modelo do cliente afeta a acurácia do modelo global. Por fim, a Seção 4.3 discute os principais achados sobre ganhos e perdas da quantização observados nos experimentos.

## 4.1 Análise de custo computacional

Esta seção apresenta os resultados da quantização para as métricas que envolvem custos computacionais. Os resultados são apresentados em seções separadas por métrica: tempo de propagação do cliente (Seção 4.1.1); consumo de memória do cliente (Seção 4.1.2); tamanho do modelo do cliente (Seção 4.1.3); e consumo de rede (Seção 4.1.4).

### *4.1.1 Tempo de propagação do cliente*

A Tabela 4 apresenta os resultados de tempo de propagação do cliente coletados em 30 repetições dos experimentos. A coluna *speedup* compara o ganho em tempo obtido com a quantização em relação ao modelo de precisão completa.

Tabela 4 – Resultados obtidos para tempo de propagação do cliente.

| Modelo / Camada de corte | Precisão do modelo | Tempo de propagação médio do cliente (s) | Speedup |
|---|---|---|---|
| VGG11 S=1 | Completa | $16,93_{\pm 0,81}$ | $4,7$ |
| | Quantizada | $3,57_{\pm 1,01}$ | |
| VGG11 S=2 | Completa | $34.01_{\pm 3,71}$ | $3,7$ |
| | Quantizada | $9.22_{\pm 1,84}$ | |
| ResNet18 S=1 | Completa | $12,63_{\pm 0,96}$ | $4,1$ |
| | Quantizada | $3,10_{\pm 1,83}$ | |
| ResNet18 S=2 | Completa | $17,36_{\pm 2,69}$ | $2,8$ |
| | Quantizada | $6,19_{\pm 0,21}$ | |
| MobileNetV2 S=1 | Completa | $4,11_{\pm 0,72}$ | $4,9$ |
| | Quantizada | $0,82_{\pm 0,15}$ | |
| MobileNetV2 S=2 | Completa | $14,01_{\pm 0,49}$ | $5,3$ |
| | Quantizada | $2,62_{\pm 0,11}$ | |

Fonte: Elaborado pelo autor.

Os resultados demonstram que houve um ganho considerável ao quantizar o modelo do cliente, com uma aceleração média de 4,3, variando entre 2,8 e 5,3. Observa-se também que o ganho no *speedup* depende da arquitetura e da escolha da camada de corte. A escolha de camada de corte mais rasa ($S = 1$) foi consistente e obteve uma aceleração superior a 4 para todos os modelos. Enquanto a camada de corte mais profunda ($S = 2$) obteve um melhor resultado apenas na MobileNetV2 (5,3), mas resultou em um *speedup* menor para a ResNet18 (2,8) e a VGG11 (3,7).

Em ambas as arquiteturas VGG11 e ResNet18, o *speedup* foi reduzido ao se utilizar a camada de corte mais profunda ($S = 2$) ao invés da camada mais rasa ($S = 1$). Nesses modelos, a principal diferença entre as camadas iniciais e as mais profundas é que, enquanto as iniciais operam sobre mapas de características com resolução alta, as camadas mais profundas processam mapas de menor resolução, pois sofrem uma redução de altura e comprimento e um aumento no número de canais após operações de *pooling*. Essa diferença afeta o tipo de computação que a camada realiza e, consequentemente, faz com que o *speedup* do primeiro bloco seja maior do que o do segundo. Este resultado é um forte indicador de que, em redes neurais que utilizam camadas convolucionais tradicionais, a quantização se torna menos eficiente ao se aprofundar a camada de corte.

Por outro lado, nota-se o comportamento oposto na arquitetura MobileNetV2, onde o *speedup* obteve um aumento de 0,4 ao se aprofundar a camada de corte. Essa arquitetura, diferente das anteriores, não se utiliza de camadas de convolução tradicionais. Ao invés disso, ela introduz o uso de convoluções separáveis em profundidade. Essa diferença de arquitetura demonstrou-se mais apropriada para manter o *speedup* da quantização, mesmo com a camada de corte mais profunda.

### *4.1.2 Consumo de memória do cliente*

A Tabela 5 aponta os resultados de média e desvio padrão obtidos ao medir o pico de memória nas 30 repetições dos experimentos.

Nota-se que a arquitetura mais densa, a VGG11, obteve uma maior redução no consumo de memória ao ter seu modelo do cliente quantizado, diminuindo em $53,3\%$ para $S = 1$ e $55,7\%$ para $S = 2$. As demais redes neurais, mais compactas, obtiveram uma redução entre $26,3\%$ e $31,5\%$.

Tabela 5 – Resultados obtidos para consumo de memória do cliente.

| Modelo / Camada de corte | Precisão do modelo | Consumo de memória do cliente (megabytes) | Redução |
|---|---|---|---|
| VGG11 S=1 | Completa | $2560.43_{\pm 1.50}$ | 53,3% |
| | Quantizada | $1289.93_{\pm 1.24}$ | |
| VGG11 S=2 | Completa | $2749.99_{\pm 1.28}$ | 55,7% |
| | Quantizada | $1129.24_{\pm 0.92}$ | |
| ResNet18 S=1 | Completa | $1299.84_{\pm 1.41}$ | 30,0% |
| | Quantizada | $909.86_{\pm 0.11}$ | |
| ResNet18 S=2 | Completa | $1390.87_{\pm 15.64}$ | 27,6% |
| | Quantizada | $1007.19_{\pm 72.41}$ | |
| MobileNetV2 S=1 | Completa | $1480.38_{\pm 1.30}$ | 26,3% |
| | Quantizada | $1090.54_{\pm 0.42}$ | |
| MobileNetV2 S=2 | Completa | $1782.11_{\pm 3.40}$ | 31,5% |
| | Quantizada | $1221.30_{\pm 0.09}$ | |

Fonte: Elaborado pelo autor.

Tabela 6 – Resultados obtidos para tamanho do modelo do cliente.

| Modelo / Camada de corte | Precisão do modelo | Tamanho do modelo do cliente (bytes) | Redução |
|---|---|---|---|
| VGG11 S=1 | Completa | 10641 | 45% |
| | Quantizada | 5774 | |
| VGG11 S=2 | Completa | 310397 | 72% |
| | Quantizada | 83956 | |
| ResNet18 S=1 | Completa | 642613 | 72% |
| | Quantizada | 174859 | |
| ResNet18 S=2 | Completa | 2760026 | 74% |
| | Quantizada | 722807 | |
| MobileNetV2 S=1 | Completa | 14133 | 30% |
| | Quantizada | 9932 | |
| MobileNetV2 S=2 | Completa | 86424 | 47% |
| | Quantizada | 45675 | |

Fonte: Elaborado pelo autor.

### *4.1.3 Tamanho do modelo do cliente*

A Tabela 6 mostra os resultados obtidos ao medir o tamanho do modelo do cliente durante a execução dos experimentos. A redução do tamanho do modelo tem um limite superior teórico de 75%, devido a razão entre o tamanho da representação na precisão completa (32 bits) a representação da versão quantizada (8 bits) do modelo. Porém, este limite não é alcançado na prática devido a uma sobrecarga vindo da quantização, que é composta por elementos não quantizáveis, como os metadados da arquitetura, e pela adição de novos parâmetros de quantização (escala e ponto-zero), necessários para a dequantização.

Tabela 7 – Resultados obtidos para consumo de rede.

| Modelo / Camada de corte | Precisão do modelo | Consumo de rede (megabytes) | Redução |
|---|---|---|---|
| VGG11 S=1 | Completa | 196,00 | 75% |
| | Quantizada | 49,00 | |
| VGG11 S=2 | Completa | 98,00 | 75% |
| | Quantizada | 24,50 | |
| ResNet18 S=1 | Completa | 49,00 | 75% |
| | Quantizada | 12,25 | |
| ResNet18 S=2 | Completa | 24,50 | 75% |
| | Quantizada | 6,12 | |
| MobileNetV2 S=1 | Completa | 49,00 | 75% |
| | Quantizada | 12,25 | |
| MobileNetV2 S=2 | Completa | 18,37 | 75% |
| | Quantizada | 4,59 | |

Fonte: Elaborado pelo autor.

Em casos onde o tamanho original do modelo do cliente ultrapassou 300000 *bytes*, a sobrecarga foi pequena e a redução se manteve entre 72% e 74%. Porém, em casos onde o tamanho do modelo do cliente é inferior a 20000 *bytes*, essa sobrecarga foi impactante e limitou a redução de tamanho para valores entre 30% e 45%. Isso evidencia que o benefício da quantização sobre o tamanho do modelo do cliente pode ser limitado em arquiteturas ou camadas de corte pequenos.

### *4.1.4 Consumo de rede*

A Tabela 7 apresenta o consumo de rede entre cliente e servidor durante a inferência. Em todos os casos, a quantização reduziu o consumo de rede em aproximadamente 75%, sendo um resultado próximo do ótimo teórico. O consumo é dominado pelo envio da IR que, por sua vez, também é quantizado junto ao modelo do cliente. Apesar da quantização do IR também incluir uma sobrecarga para os parâmetros de escala e ponto-zero, seu impacto é baixo em comparação com o tamanho total do tensor.

## 4.2 Análise de acurácia

Esta seção se destina a apresentar os efeitos da quantização do modelo do cliente sobre a acurácia do modelo global. Os resultados observados foram obtidos após 10 repetições do treino do modelo. Os resultados são apresentados para cada modelo.

### *4.2.1 VGG11*

A Tabela 8 apresenta os resultados de acurácia do modelo com arquitetura VGG11, obtidos após o treinamento e a aplicação dos esquemas de quantização. Como pode ser observado, a acurácia do VGG11 se manteve entre 91% e 92% em todas as configurações. A diferença entre o modelo com precisão completa e as versões quantizadas se mostrou pouco significativa. Mesmo em um caso mais desafiador, com duas camadas quantizadas ($S = 2$), dados não-IID e quantização de 8 bits, o modelo obteve a acurácia de 91%. Isso evidencia que a acurácia do VGG11 não é apenas robusta à quantização no cliente, mas também resistente a condições de distribuição de dados não-IID no cenário de ADF.

Tabela 8 – Resultados para acurácia do modelo VGG11.

| Modelo / Camada de corte | Conjunto de dados | Quantização | Acurácia | Diferença relativa precisão completa |
|---|---|---|---|---|
| VGG11 S=1 | CIFAR-10 IID N=4 | Precisão completa | $91,7\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $91,7\%_{\pm 0.2}$ | $0,0\%$ |
| | | TCQ | $91,7\%_{\pm 0.1}$ | $-0,1\%$ |
| | CIFAR-10 não-IID N=4 | Precisão completa | $91,9\%_{\pm 0.1}$ | $0,0\%$ |
| | | QPT | $91,9\%_{\pm 0.1}$ | $0,0\%$ |
| | | TCQ | $91,8\%_{\pm 0.1}$ | $-0,1\%$ |
| | CIFAR-10 IID N=8 | Precisão completa | $91,8\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $91,8\%_{\pm 0.2}$ | $0,0\%$ |
| | | TCQ | $91,6\%_{\pm 0.1}$ | $-0,2\%$ |
| | CIFAR-10 não-IID N=8 | Precisão completa | $91,3\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $91,3\%_{\pm 0.2}$ | $0,0\%$ |
| | | TCQ | $91,2\%_{\pm 0.2}$ | $-0,1\%$ |
| VGG11 S=2 | CIFAR-10 IID N=4 | Precisão completa | $91,7\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $91,7\%_{\pm 0.2}$ | $0,0\%$ |
| | | TCQ | $91,9\%_{\pm 0.1}$ | $0,2\%$ |
| | CIFAR-10 não-IID N=4 | Precisão completa | $91,6\%_{\pm 0.1}$ | $0,0\%$ |
| | | QPT | $91,6\%_{\pm 0.1}$ | $0,0\%$ |
| | | TCQ | $91,5\%_{\pm 0.2}$ | $-0,1\%$ |
| | CIFAR-10 IID N=8 | Precisão completa | $91,6\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $91,6\%_{\pm 0.1}$ | $0,0\%$ |
| | | TCQ | $91,6\%_{\pm 0.1}$ | $0,0\%$ |
| | CIFAR-10 não-IID N=8 | Precisão completa | $91,2\%_{\pm 0.1}$ | $0,0\%$ |
| | | QPT | $91,2\%_{\pm 0.1}$ | $0,0\%$ |
| | | TCQ | $91,1\%_{\pm 0.1}$ | $-0,1\%$ |

Fonte: Elaborado pelo autor.

### *4.2.2 ResNet18*

A Tabela 9 apresenta os resultados de acurácia obtidos pelo modelo com a arquitetura ResNet18, em que a acurácia se mostrou menos estável, variando entre 76% e 92%. A acurácia

Tabela 9 – Resultados obtidos para acurácia do modelo ResNet18.

| Modelo / Camada de corte | Conjunto de dados | Quantização | Acurácia | Diferença relativa precisão completa |
|---|---|---|---|---|
| ResNet18 S=1 | CIFAR-10 IID N=4 | Precisão completa | $92,5\%_{\pm 0.1}$ | $0,0\%$ |
| | | QPT | $92,4\%_{\pm 0.1}$ | $-0,2\%$ |
| | | TCQ | $92,2\%_{\pm 0.1}$ | $-0,3\%$ |
| | CIFAR-10 não-IID N=4 | Precisão completa | $81,2\%_{\pm 0.6}$ | $0,0\%$ |
| | | QPT | $80,7\%_{\pm 0.6}$ | $-0,5\%$ |
| | | TCQ | $81,3\%_{\pm 0.9}$ | $0,1\%$ |
| | CIFAR-10 IID N=8 | Precisão completa | $92,6\%_{\pm 0.1}$ | $0,0\%$ |
| | | QPT | $92,5\%_{\pm 0.2}$ | $-0,1\%$ |
| | | TCQ | $92,7\%_{\pm 0.1}$ | $0,1\%$ |
| | CIFAR-10 não-IID N=8 | Precisão completa | $78,4\%_{\pm 0.7}$ | $0,0\%$ |
| | | QPT | $78,4\%_{\pm 0.7}$ | $0,0\%$ |
| | | TCQ | $78,2\%_{\pm 1.0}$ | $-0,2\%$ |
| ResNet18 S=2 | CIFAR-10 IID N=4 | Precisão completa | $92,4\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $92,3\%_{\pm 0.2}$ | $-0,1\%$ |
| | | TCQ | $92,4\%_{\pm 0.1}$ | $0,0\%$ |
| | CIFAR-10 não-IID N=4 | Precisão completa | $80,8\%_{\pm 0.7}$ | $0,0\%$ |
| | | QPT | $80,4\%_{\pm 0.6}$ | $-0,4\%$ |
| | | TCQ | $80,6\%_{\pm 1.0}$ | $-0,2\%$ |
| | CIFAR-10 IID N=8 | Precisão completa | $92,5\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $92,5\%_{\pm 0.2}$ | $-0,1\%$ |
| | | TCQ | $92,4\%_{\pm 0.1}$ | $-0,1\%$ |
| | CIFAR-10 não-IID N=8 | Precisão completa | $76,6\%_{\pm 1.0}$ | $0,0\%$ |
| | | QPT | $76,5\%_{\pm 1.1}$ | $0,1\%$ |
| | | TCQ | $77,0\%_{\pm 0.9}$ | $0,3\%$ |

Fonte: Elaborado pelo autor.

no pior caso, de 76%, ocorreu quando a rede neural foi dividida com a camada de corte $S = 2$ e treinada com dados não-IID distribuídos entre 8 clientes. Isso demonstra que, em comparação com a arquitetura VGG11, o modelo é mais sensível às condições hostis de cenários ADF. Uma possível explicação está na própria eficiência da ResNet18, que por ser uma arquitetura mais compacta e com menos redundância que a VGG11, tornou-se também mais volátil à heterogeneidade na distribuição dos dados.

Houve uma redução da acurácia devido à distribuição dos dados não-IID, mas o impacto da quantização do modelo do cliente, por si só, não reduz tanto a acurácia da solução. Em todas as configurações, a maior perda de acurácia observada foi de $0,5\%$ ao aplicar QPT. Mesmo quando o modelo do cliente foi dividido na camada de corte mais profunda, $S = 2$, a perda de qualidade não foi de grande impacto. Isso demonstra que, apesar da arquitetura da ResNet18 ser mais compacta do que a da VGG11, ela ainda possui redundâncias que podem ser exploradas. Ademais, não foi observada diferença significativa entre as técnicas QPT e TCQ.

Tabela 10 – Resultados obtidos para acurácia do modelo MobileNetV2.

| Modelo / Camada de corte | Conjunto de dados | Quantização | Acurácia | Diferença relativa precisão completa |
|---|---|---|---|---|
| MobileNetV2 S=1 | CIFAR-10 IID N=4 | Precisão completa | $88,2\%_{\pm 0.3}$ | $0,0\%$ |
| | | QPT | $84,0\%_{\pm 3.1}$ | $-4,2\%$ |
| | | TCQ | $87,1\%_{\pm 0.3}$ | $-1,1\%$ |
| | CIFAR-10 não-IID N=4 | Precisão completa | $73,4\%_{\pm 0.3}$ | $0,0\%$ |
| | | QPT | $66,0\%_{\pm 4.6}$ | $-7,4\%$ |
| | | TCQ | $70,8\%_{\pm 0.2}$ | $-2,6\%$ |
| | CIFAR-10 IID N=8 | Precisão completa | $88,6\%_{\pm 0.3}$ | $0,0\%$ |
| | | QPT | $85,2\%_{\pm 1.4}$ | $-3,4\%$ |
| | | TCQ | $87,6\%_{\pm 0.4}$ | $-1,0\%$ |
| | CIFAR-10 não-IID N=8 | Precisão completa | $67,3\%_{\pm 0.6}$ | $0,0\%$ |
| | | QPT | $65,8\%_{\pm 1.3}$ | $-1,5\%$ |
| | | TCQ | $65,0\%_{\pm 0.5}$ | $-2,3\%$ |
| MobileNetV2 S=2 | CIFAR-10 IID N=4 | Precisão completa | $88,1\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $77,7\%_{\pm 2.9}$ | $-10,4\%$ |
| | | TCQ | $87,2\%_{\pm 0.1}$ | $-0,9\%$ |
| | CIFAR-10 não-IID N=4 | Precisão completa | $72,7\%_{\pm 1.3}$ | $0,0\%$ |
| | | QPT | $59,9\%_{\pm 2.3}$ | $-12,8\%$ |
| | | TCQ | $70,3\%_{\pm 1.9}$ | $-2,4\%$ |
| | CIFAR-10 IID N=8 | Precisão completa | $88,5\%_{\pm 0.2}$ | $0,0\%$ |
| | | QPT | $81,9\%_{\pm 2.3}$ | $-6,6\%$ |
| | | TCQ | $87,3\%_{\pm 0.3}$ | $-1,2\%$ |
| | CIFAR-10 não-IID N=8 | Precisão completa | $64,8\%_{\pm 2.6}$ | $0,0\%$ |
| | | QPT | $39,0\%_{\pm 0.5}$ | $-22,7\%$ |
| | | TCQ | $60,6\%_{\pm 1.1}$ | $-2,8\%$ |

Fonte: Elaborado pelo autor.

### *4.2.3 MobileNetV2*

Por fim, a Tabela 10 aponta os resultados de acurácia coletados após o treinamento e a quantização do modelo com arquitetura MobileNetV2.

Dentre os três modelos avaliados, o MobileNetV2 obteve maior variabilidade, com resultados na faixa de 39% e 88%. A distribuição não-IID dos dados teve grande impacto na acurácia, especialmente quando os dados são distribuídos com um número maior de clientes ($N = 8$). Nesses casos, a acurácia dos modelos com precisão completa foi reduzida em mais de 20% em relação à acurácia dos modelos com configurações semelhantes, mas treinados com dados IID. Isso evidencia que a arquitetura MobileNetV2 é mais sensível à distribuições de dados heterogêneas, comuns em cenários de ADF. Por ser o modelo mais compacto e otimizado para *hardware* dentre os listados, é esperado que o modelo MobileNetV2 seja mais eficiente computacionalmente, mas essa eficiência vem ao custo de uma menor robustez devido a pouca ou nenhuma redundância nos parâmetros da rede.

Algo semelhante ocorre ao se observar a redução da acurácia causada pela quantiza-

ção. Ao aplicar a quantização com a camada de corte rasa $S = 1$, o impacto causado alcança um máximo de $7,4\%$. Observa-se então que o modelo possui resistência moderada à quantização quando aplicada apenas nas camadas iniciais.

No entando, esse cenário muda ao aprofundar a camada de corte para $S = 2$. Nota-se uma redução drástica na acurácia ao se utilizar QPT, resultando em uma perda de $22,7\%$ quando os dados são distribuídos de forma não-IID em 8 clientes, tornando a solução inviável. Essa perda de acurácia é mitigada quando o TCQ é aplicado, alcançando uma redução na acurácia limitada a no máximo $2,8\%$ em todos os casos. Isso demonstra que, para modelos mais otimizados computacionalmente, a computação extra de ajuste fino realizada pelo TCQ pode superar o QPT com uma margem alta.

## 4.3 Discussão

Os experimentos mostraram que a quantização é uma ferramenta poderosa para otimizar a computação em soluções de ADF, mas não é uma solução universal. Os ganhos de eficiência e a perda de acurácia dependem da arquitetura da rede neural, distribuição dos dados e esquema de quantização.

Os resultados indicam que arquiteturas mais densas e com maior redundância são mais resistentes ao ruído causado pela quantização do modelo do cliente. Os modelos *VGG11* e *ResNet18* obtiveram a perda da acurácia do modelo menor que 2% em todos os casos. Nesses casos, a escolha entre QPT e TCQ não apresentou diferença notável.

Por outro lado, a arquitetura *MobileNetV2*, mais compacta, foi mais sensível e apresentou uma perda de 22,7% de acurácia no pior caso ($S = 2$, não-IID e 8 clientes) ao utilizar o esquema QPT. Esse caso mostrou a importância do TCQ que permitiu uma perda na acurácia de apenas 2,8%. Portanto, o TCQ mostrou-se eficaz na quantização de arquiteturas compactas, mesmo quando apenas o modelo do cliente é quantizado em soluções de ADF, o que justifica o custo adicional de retreino.

O *speedup* é outra métrica que variou conforme a arquitetura. Nas redes *VGG11* e *ResNet18* o *speedup* diminuiu ao se aprofundar a camada de corte de $S = 1$ para $S = 2$, enquanto a *MobileNetV2* ele aumentou, indicando que as camadas com convoluções separáveis em profundidade se comportam diferentes de forma diferente à quantização em relação às camadas convolucionais tradicionais.

Ao cruzar os dados de acurácia com o tempo de propagação do cliente, o caso mais

extremo foi com a rede *MobileNetV2* ($S = 2$), que alcançou o maior *speedup* (5,3) de todas configurações , mas também apresentou a maior perda de acurácia, de 22,7% com QPT e 2,8% com TCQ. Em contrapartida, o modelo *ResNet18* ($S = 2$) apresentou o menor *speedup* de 2,8, mas obteve baixa degradação na acurácia (abaixo de 1%) em ambos os QPT e TCQ. Isso mostra que a relação de ganhos e perdas não é trivial e está ligada à arquitetura e distribuição de dados. Um modelo mais compactado tem tempo de propagação menor e é mais acelerado ao aplicar a quantização, porém é mais sensível à perda de acurácia, especialmente ao aplicar QPT.

Em relação ao pico de memória, a arquitetura *VGG11* apresentou uma redução maior do que os demais modelos após a quantização do modelo do cliente, alcançando uma média de 54%, enquanto as demais arquiteturas obtiveram uma média de 29%. Isso se explica porque o armazenamento do mapa de características gerado na inferência (GHOLAMI *et al.*, 2022) domina o custo de memória em camadas convolucionais, e *VGG11* realiza as operações de convolução sobre a imagem na resolução original de $224\times224$ ainda em suas primeiras camadas. Enquanto isso, as demais arquiteturas reduzem essa resolução para $112\times112$, com *pooling* das próprias convoluções. Logo, como o mapa de características no *VGG11* é proporcionalmente maior do que o dos outros modelos, a sua quantização de 32 bits para 8 bits resulta em uma economia de memória maior.

A redução no consumo de rede foi previsível, mantendo-se próxima do ótimo teórico de 75% em todos os casos, pois consiste na redução direta do IR, que é um simples vetor numérico. Por outro lado, a redução do tamanho do modelo do cliente variou entre 30% a 72%. Em casos onde a partição do modelo do cliente possui é pequena, a sobrecarga da própria estrutura da rede neural e da quantização é relevante.

Em suma, a análise revelou que redes com maior redundância são mais resistentes, suportando o ruído da quantização e permitindo esquemas mais simples como o QPT. Por outro lado, arquiteturas compactas e já otimizadas demonstraram ser sensíveis, exigindo o custo computacional adicional do TCQ para evitar perdas drásticas de acurácia. Demonstrou-se que os ganhos de eficiência não são lineares e dependem de características específicas de cada arquitetura, como o tipo de convolução ou o processamento da resolução do mapa de características. Isso mostra que o projeto de soluções de ADF deve considerar cuidadosamente esses fatores para manter o balanço entre ganho computacional e acurácia do modelo ao se aplicar a quantização do modelo do cliente.

# 5 CONCLUSÃO

Este trabalho teve como objetivo investigar o uso de quantização para mitigar os custos computacionais da inferência no lado do cliente em soluções de ADF. Para isso, foi realizada uma análise de desempenho empírica considerando diferentes configurações para medir ganhos de eficiência em tempo, memória, rede e o impacto negativo na acurácia.

A análise confirmou ganhos de eficiência da quantização, mas é observado também que esse ganho varia de acordo com a arquitetura da rede neural e camada de corte. A quantização do modelo do cliente trouxe um *speedup* médio de 4,3 no tempo de propagação. Ao mesmo tempo, o *speedup* nas redes *VGG11* e *ResNet18* diminui à medida que se aprofunda a camada de corte. Isso mostra que a arquitetura das camadas convolucionais iniciais desses modelos são mais aptas para manter o *speedup* da quantização. Por outro lado, as convoluções separáveis em profundidade da rede *MobileNet* mantiveram o *speedup* de 4,9 para $S = 1$ e de 5,3 para $S = 2$, mesmo com uma camada de corte mais profunda. Isso evidencia que a escolha da camada de corte deve considerar tanto a capacidade computacional do cliente como as propriedades da arquitetura da rede.

O consumo de memória do cliente foi reduzido de forma mais efetiva na rede neural *VGG11*, com uma média de redução de 54%, possivelmente devido a essa rede também consumir mais espaço na memória que as demais, que obtiveram uma redução média em torno de 31%.

A redução no consumo de rede se manteve próximo do ótimo de 75% para todos os casos, o que é esperado devido a quantização do IR de 32 bits para 8 bits. Por outro lado, a redução do tamanho do modelo do cliente variou entre 30% a 72%. Foi concluído que fatores de arquitetura e escolha da camada de corte afetam a eficiência da quantização na redução do tamanho do modelo do cliente: modelos maiores obtiveram uma redução mais próxima do ótimo de 75%, enquanto a sobrecarga da quantização impacta mais os modelos pequenos.

Os resultados de acurácia variam conforme a arquitetura do modelo. Um modelo com muita redundância, como o *VGG11* e *Resnet18*, é capaz de ser quantizado com QPT ou TCQ sem perda significante em sua acurácia (abaixo de 2%). Enquanto um modelo com arquitetura projetada para ser mais compacta, como o *MobileNetV2*, tende a tomar um impacto maior após ser quantizado. Nesse caso, a aplicação do QPT resultou em uma perda de acurácia de 22,7%, enquanto o uso de TCQ manteve uma perda de apenas 2,8%. Isso mostra que a economia de se usar QPT é viável em modelos mais robustos, enquanto arquiteturas para dispositivos leves requerem o uso de TCQ.

A principal contribuição desse trabalho é a evidência empírica de que a quantização não é uma solução universal, mas seus ganhos dependem da interação complexa de diversas configurações como a arquitetura do modelo, a distribuição dos dados e o esquema de quantização. Dessa forma, essa dissertação serve como um guia preliminar para auxiliar pesquisadores e engenheiros no desenvolvimento de soluções de ADF.

## 5.1 Limitações e trabalhos futuros

A análise foi limitada a arquiteturas de redes convolucionais e a conjuntos de dados de classificação de imagens. Além disso o ambiente simulado em nuvem não representa realisticamente as limitações de *hardware* reais, mesmo que utilizando máquinas virtuais limitadas.

Como trabalhos futuros, sugere-se:

- Explorar problemas de outros domínios de aprendizado de máquina. Por exemplo, segmentação de imagens, séries temporais e processamento de linguagem natural.
- Analisar o impacto da quantização em ambientes móveis e IoT com aplicações de tempo real, analisando também a métrica de consumo de energia.
- Explorar níveis mais agressivos de quantização, como 4 bits, 2 bits, 1 bit ou precisão mista.
- Explorar como a quantização afeta a privacidade dos dados transmitidos pelo cliente para o servidor.

# REFERÊNCIAS

ALOM, M. Z.; TAHA, T. M.; YAKOPCIC, C.; WESTBERG, S.; SIDIKE, P.; NASRIN, M. S.; HASAN, M.; ESSEN, B. C. V.; AWWAL, A. A.; ASARI, V. K. A state-of-the-art survey on deep learning theory and architectures. **Electronics**, Multidisciplinary Digital Publishing Institute, v. 8, n. 3, p. 292, 2019.

ANSEL, J.; YANG, E.; HE, H.; GIMELSHEIN, N.; JAIN, A.; VOZNESENSKY, M.; BAO, B.; BELL, P.; BERARD, D.; BUROVSKI, E.; CHAUHAN, G.; CHOURDIA, A.; CONSTABLE, W.; DESMAISON, A.; DEVITO, Z.; ELLISON, E.; FENG, W.; GONG, J.; GSCHWIND, M.; HIRSH, B.; HUANG, S.; KALAMBARKAR, K.; KIRSCH, L.; LAZOS, M.; LEZCANO, M.; LIANG, Y.; LIANG, J.; LU, Y.; LUK, C.; MAHER, B.; PAN, Y.; PUHRSCH, C.; RESO, M.; SAROUFIM, M.; SIRAICHI, M. Y.; SUK, H.; SUO, M.; TILLET, P.; WANG, E.; WANG, X.; WEN, W.; ZHANG, S.; ZHAO, X.; ZHOU, K.; ZOU, R.; MATHEWS, A.; CHANAN, G.; WU, P.; CHINTALA, S. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. *In*: **29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)**. ACM, 2024. Disponível em: https://pytorch.org/assets/pytorch2-2.pdf. Acesso em: 26 set. 2025.

ANSEL, J.; YANG, E.; HE, H.; GIMELSHEIN, N.; JAIN, A.; VOZNESENSKY, M.; BAO, B.; BELL, P.; BERARD, D.; BUROVSKI, E.; CHAUHAN, G.; CHOURDIA, A.; CONSTABLE, W.; DESMAISON, A.; DEVITO, Z.; ELLISON, E.; FENG, W.; GONG, J.; GSCHWIND, M.; HIRSH, B.; HUANG, S.; KALAMBARKAR, K.; KIRSCH, L.; LAZOS, M.; LEZCANO, M.; LIANG, Y.; LIANG, J.; LU, Y.; LUK, C.; MAHER, B.; PAN, Y.; PUHRSCH, C.; RESO, M.; SAROUFIM, M.; SIRAICHI, M. Y.; SUK, H.; SUO, M.; TILLET, P.; WANG, E.; WANG, X.; WEN, W.; ZHANG, S.; ZHAO, X.; ZHOU, K.; ZOU, R.; MATHEWS, A.; CHANAN, G.; WU, P.; CHINTALA, S. **Pytorch Documentation**: Introduction to quantization. 2024. Disponível em: https://pytorch.org/docs/stable/quantization.html. Acesso em: 14 out. 2025.

AYAD, A.; BARHOUSH, M.; LOH, J.; GEMMEKE, T.; SCHMEINK, A. Peace: Private and energy-efficient algorithm for cardiac evaluation on the edge using modified split learning and model quantization. *In*: **2023 14th International Conference on Information and Communication Systems (ICICS)**. [*S. l.*: *s. n.*], 2023. p. 01–06.

BURKOV, A. **The Hundred-Page Machine Learning Book**. Andriy Burkov, 2019. ISBN 9781999579517. Disponível em: https://books.google.com.br/books?id=0jbxwQEACAAJ. Acesso em: 23 jul. 2024.

CHEN, X.; LI, J.; CHAKRABARTI, C. Communication and computation reduction for split learning using asynchronous training. *In*: **2021 IEEE Workshop on Signal Processing Systems (SiPS)**. [*S. l.*: *s. n.*], 2021. p. 76–81.

CHOLLET, F. **Deep Learning with Python**. Second. Manning, 2021. ISBN 9781617296864. Disponível em: https://www.manning.com/books/deep-learning-with-python-second-edition?a_aid=keras. Acesso em: 26 set. 2025.

DACHILLE, J.; HUANG, C.; LIU, X. **The Impact of Cut Layer Selection in Split Federated Learning**. 2024. Disponível em: https://arxiv.org/abs/2412.15536. Acesso em: 12 dez. 2025.

DUAN, J.; DUAN, J.; WAN, X.; LI, Y. Efficient federated learning method for cloud-edge network communication. *In*: **2023 5th International Conference on Communications, Information System and Computer Engineering (CISCE)**. [*S. l.*: *s. n.*], 2023. p. 118–121.

G., D. M. J.; SOLANS, D.; HEIKKILA, M.; VITALETTI, A.; KOURTELLIS, N.; ANAGNOSTOPOULOS, A.; CHATZIGIANNAKIS, I. **Non-IID data in Federated Learning**: A survey with taxonomy, metrics, methods, frameworks and future directions. 2024. Disponível em: https://arxiv.org/abs/2411.12377. Acesso em: 18 jan. 2026.

GAO, Y.; KIM, M.; ABUADBBA, S.; KIM, Y.; THAPA, C.; KIM, K.; CAMTEP, S.; KIM, H.; NEPAL, S. End-to-end evaluation of federated learning and split learning for internet of things. *In*: . [*S. l.*: *s. n.*], 2020. p. 91–100.

GHOLAMI, A.; KIM, S.; DONG, Z.; YAO, Z.; MAHONEY, M. W.; KEUTZER, K. A survey of quantization methods for efficient neural network inference. *In*: **Low-Power Computer Vision**. [*S. l.*]: Chapman and Hall/CRC, 2022. p. 291–326.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S. l.*]: MIT Press, 2016. http://www.deeplearningbook.org. Acesso em: 12 jul. 2024.

GUPTA, A.; RITWIK; CHAURASIYA, V. K. Adaptive low-latency split federated learning with dynamic model partitioning in resource-constrained healthcare iot. **IEEE Transactions on Green Communications and Networking**, v. 10, p. 1308–1321, 2026.

GUPTA, O.; RASKAR, R. Distributed learning of deep neural network over multiple agents. **Journal of Network and Computer Applications**, v. 116, p. 1–8, 08 2018.

HAFI, H.; BRIK, B.; FRANGOUDIS, P. A.; KSENTINI, A.; BAGAA, M. Split federated learning for 6g enabled-networks: Requirements, challenges, and future directions. **IEEE Access**, v. 12, p. 9890–9930, 2024.

HAYKIN, S. **Neural Networks and Learning Machines**. [*S. l.*]: Prentice Hall, 2009. (Neural networks and learning machines, v. 10). ISBN 9780131471399.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. *In*: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [*S. l.*: *s. n.*], 2016. p. 770–778.

HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. **MobileNets**: Efficient convolutional neural networks for mobile vision applications. 2017. Disponível em: https://arxiv.org/abs/1704.04861. Acesso em: 26 set. 2025.

IEEE Standard for Floating-Point Arithmetic. **IEEE Std 754-2019 (Revision of IEEE 754-2008)**, p. 1–84, 2019.

JOSHI, P.; HASANUZZAMAN, M.; THAPA, C.; AFLI, H.; SCULLY, T. Enabling all in-edge deep learning: A literature review. **IEEE Access**, v. 11, p. 3431–3460, 2023.

KAIROUZ, P.; MCMAHAN, H. B.; AVENT, B.; BELLET, A.; BENNIS, M.; BHAGOJI, A. N.; BONAWIT, K.; CHARLES, Z.; CORMODE, G.; CUMMINGS, R.; D'OLIVEIRA, R. G. L.; EICHNER, H.; ROUAYHEB, S. E.; EVANS, D.; GARDNER, J.; GARRETT, Z.; GASCóN, A.; GHAZI, B.; GIBBONS, P. B.; GRUTESER, M.; HARCHAOUI, Z.; HE, C.; HE, L.; HUO, Z.;

HUTCHINSON, B.; HSU, J.; JAGGI, M.; JAVIDI, T.; JOSHI, G.; KHODAK, M.; KONECNý, J.; KOROLOVA, A.; KOUSHANFAR, F.; KOYEJO, S.; LEPOINT, T.; LIU, Y.; MITTAL, P.; MOHRI, M.; NOCK, R.; ÖZGüR, A.; PAGH, R.; QI, H.; RAMAGE, D.; RASKAR, R.; RAYKOVA, M.; SONG, D.; SONG, W.; STICH, S. U.; SUN, Z.; SURESH, A. T.; TRAMèR, F.; VEPAKOMMA, P.; WANG, J.; XIONG, L.; XU, Z.; YANG, Q.; YU, F. X.; YU, H.; ZHAO, S. [*S. l.*: *s. n.*], 2021.

KRISHNAMOORTHI, R. **Quantizing deep convolutional networks for efficient inference**: A whitepaper. 2018. Disponível em: https://arxiv.org/abs/1806.08342. Acesso em: 26 set. 2025.

KRIZHEVSKY, A.; HINTON, G. **Learning multiple layers of features from tiny images**. Toronto, Ontario, 2009. Disponível em: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. Acesso em: 02 fev. 2025.

KULKARNI, U.; HOSAMANI, A. S.; MASUR, A. S.; HEGDE, S.; VERNEKAR, G. R.; CHANDANA, K. S. A survey on quantization methods for optimization of deep neural networks. *In*: **2022 International Conference on Automation, Computing and Renewable Systems (ICACRS)**. [*S. l.*: *s. n.*], 2022. p. 827–834.

KUMAR, S. K. On weight initialization in deep neural networks. **CoRR**, abs/1704.08863, 2017. Disponível em: http://arxiv.org/abs/1704.08863. Acesso em: 25 set. 2025.

LI, L.; FAN, Y.; TSE, M.; LIN, K.-Y. A review of applications in federated learning. **Computers Industrial Engineering**, v. 149, p. 106854, 2020. ISSN 0360-8352. Disponível em: https://www.sciencedirect.com/science/article/pii/S0360835220305532. Acesso em: 08 mar. 2025.

LUDWIG, H.; BARACALDO, N. (Ed.). **Federated Learning - A Comprehensive Overview of Methods and Applications**. Springer, 2022. ISBN 978-3-030-96896-0. Disponível em: https://doi.org/10.1007/978-3-030-96896-0. Acesso em: 12 dez. 2024.

LYU, L.; YU, H.; MA, X.; CHEN, C.; SUN, L.; ZHAO, J.; YANG, Q.; YU, P. S. Privacy and robustness in federated learning: Attacks and defenses. **IEEE Transactions on Neural Networks and Learning Systems**, p. 1–21, 2022.

Lê, M. T.; WOLINSKI, P.; ARBEL, J. **Efficient Neural Networks for Tiny Machine Learning**: A comprehensive review. 2023. Disponível em: https://arxiv.org/abs/2311.11883. Acesso em: 26 set. 2025.

MCMAHAN, B.; RAMAGE, D. **Federated Learning**: Collaborative machine learning without centralized training data. 2017. Disponível em: https://blog.research.google/2017/04/federated-learning-collaborative.html. Acesso em: 15 dez. 2024.

MENGHANI, G. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 55, n. 12, mar 2023. ISSN 0360-0300. Disponível em: https://doi.org/10.1145/3578938.

MULHOLLAND, C. S. Dados pessoais sensíveis e a tutela de direitos fundamentais: uma análise à luz da lei geral de proteção de dados (lei 13.709/18). **Revista de Direitos e Garantias Fundamentais**, v. 19, n. 3, p. 159–180, dez. 2018. Disponível em: https://sisbib.emnuvens.com.br/direitosegarantias/article/view/1603. Acesso em: 26 set. 2025.

NAGEL, M.; FOURNARAKIS, M.; AMJAD, R. A.; BONDARENKO, Y.; BAALEN, M. van; BLANKEVOORT, T. **A White Paper on Neural Network Quantization**. 2021. Disponível em: https://arxiv.org/abs/2106.08295. Acesso em: 24 set. 2025.

OH, Y.; LEE, J.; BRINTON, C. G.; JEON, Y.-S. Communication-efficient split learning via adaptive feature-wise compression. **IEEE Transactions on Neural Networks and Learning Systems**, v. 36, n. 6, p. 10844–10858, 2025.

PASQUINI, D.; ATENIESE, G.; BERNASCHI, M. Unleashing the tiger: Inference attacks on split learning. *In*: **Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security**. New York, NY, USA: Association for Computing Machinery, 2021. (CCS '21), p. 2113–2129. ISBN 9781450384544. Disponível em: https://doi.org/10.1145/3460120.3485259. Acesso em: 26 set. 2025.

PHAM, N. D.; ABUADBBA, A.; GAO, Y.; PHAN, K. T.; CHILAMKURTI, N. Binarizing split learning for data privacy enhancement and computation reduction. **Trans. Info. For. Sec.**, IEEE Press, v. 18, p. 3088–3100, jan. 2023. ISSN 1556-6013. Disponível em: https://doi.org/10.1109/TIFS.2023.3274391. Acesso em: 26 set. 2025.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *In*: BENGIO, Y.; LECUN, Y. (Ed.). **3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings**. [*S. n.*], 2015. Disponível em: http://arxiv.org/abs/1409.1556. Acesso em: 26 set. 2025.

STALLINGS, W. **Arquitetura e Organização de Computadores**. Pearson Universidades, 2009. ISBN 9788576055648. Disponível em: https://books.google.com.br/books?id=kTKeQwAACAAJ. Acesso em: 26 set. 2025.

SUN, T.; LI, D.; WANG, B. **Decentralized Federated Averaging**. 2021.

TAN, A. Z.; YU, H.; CUI, L.; YANG, Q. Towards personalized federated learning. **IEEE Transactions on Neural Networks and Learning Systems**, v. 34, n. 12, p. 9587–9603, 2023.

THAPA, C.; ARACHCHIGE, P. C. M.; CAMTEPE, S.; SUN, L. Splitfed: When federated learning meets split learning. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 36, n. 8, p. 8485–8493, Jun. 2022. Disponível em: https://ojs.aaai.org/index.php/AAAI/article/view/20825. Acesso em: 26 set. 2025.

TRUONG, N.; SUN, K.; WANG, S.; GUITTON, F.; GUO, Y. Privacy preservation in federated learning: An insightful survey from the gdpr perspective. **Computers Security**, v. 110, p. 102402, 2021. ISSN 0167-4048. Disponível em: https://www.sciencedirect.com/science/article/pii/S0167404821002261. Acesso em: 22 set. 2024.

WADHWA, M.; GUPTA, G. R.; SAHU, A.; SAINI, R.; MITTAL, V. Pfsl:: Personalized fair split learning with data label privacy for thin clients. *In*: **2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)**. Los Alamitos, CA, USA: IEEE Computer Society, 2023. p. 377–390. Disponível em: https://doi.ieeecomputersociety.org/10.1109/CCGrid57682.2023.00043. Acesso em: 28 set. 2025.

WENG, O. **Neural Network Quantization for Efficient Inference**: A survey. 2023. Disponível em: https://arxiv.org/abs/2112.06126. Acesso em: 26 set. 2025.

WU, H.; JUDD, P.; ZHANG, X.; ISAEV, M.; MICIKEVICIUS, P. **Integer Quantization for Deep Learning Inference**: Principles and empirical evaluation. 2020. Disponível em: https://arxiv.org/abs/2004.09602. Acesso em: 26 set. 2025.

YEGULALP, S. **Facebook brings GPU-powered machine learning to Python**. 2017. Disponível em: https://www.infoworld.com/article/3159120/facebook-brings-gpu-powered-machine-learning-to-python.html. Acesso em: 14 out. 2025.

ZHANG, J.; NI, W.; WANG, D. Federated split learning with model pruning and gradient quantization in wireless networks. **IEEE Transactions on Vehicular Technology**, v. 74, n. 4, p. 6850–6855, 2025.

ZHANG, S.; CHENG, G.; WU, W.; HUANG, X.; SONG, L.; SHEN, X. Split fine-tuning for large language models in wireless networks. **IEEE Journal of Selected Topics in Signal Processing**, v. 19, n. 7, p. 1376–1391, 2025.