Dispositivos, Eu Escolho Vocês: Seleção de Clientes Adaptativa para Comunicação Eficiente em Aprendizado Federado

Allan M. de Souza[1], **Luiz F. Bittencourt**[1], **Eduardo Cerqueira**[2], **Antonio A. F. Loureiro**[3], **Leandro A. Villas**[1]

[1]Universidade Estadual de Campinas, Brasil
[2] Universidade Federal do Pará, Brasil
[3] Universidade Federal de Minas Gerais, Brasil

{allanms, lfb00, lvilas}@ic.unicamp.br
cerqueira@ufpa.br, loureiro@dcc.ufmg.br

Resumo. *O aprendizado federado (Federated Learning – FL) é uma abordagem distribuída para o treinamento colaborativo de modelos de aprendizado de máquina. O FL requer um alto nível de comunicação entre os dispositivos e um servidor central, assim gerando diversos desafios, incluindo gargalos de comunicação e escalabilidade na rede. Neste trabalho, introduzimos DEEV, uma solução para diminuir os custos gerais de comunicação e computação para treinar um modelo no ambiente FL. DEEV emprega uma estratégia de seleção de clientes que adapta dinamicamente o número de dispositivos que treinam o modelo e o número de rodadas necessárias para atingir a convergência. Um caso de uso no conjunto de dados de reconhecimento de atividades humanas é realizado para avaliar DEEV e compará-lo com outras abordagens do estado da arte. Avaliações experimentais mostram que DEEV reduz eficientemente a sobrecarga geral de comunicação e computação para treinar um modelo e promover sua convergência. Em particular, o DEEV reduz em até 60% a comunicação e em até 90% a sobrecarga de computação em comparação com as abordagens da literatura, ao mesmo tempo em que fornece boa convergência mesmo em cenários em que os dados são distribuídos de forma não independente e idêntica entre os dispositivos clientes.*

Abstract. *Federated Learning (FL) is a distributed approach to collaboratively training machine learning models. FL requires a high level of communication between the devices and a central server, thus creating several challenges, including communication bottlenecks and network scalability. This work introduces DEEV, a solution to lower overall communication and computation costs for training a model in the FL environment. DEEV employs a client selection strategy that dynamically adapts the number of devices training the model and the number of rounds required to achieve convergence. A use case in the human activity recognition dataset is performed to evaluate DEEV and compare it to other state-of-the-art approaches. Experimental evaluations show that DEEV efficiently reduces the overall communication and computation overhead to train a model and promote its convergence. In particular, DEEV reduces up to 60% communication and 90% computational overhead compared to literature approaches while providing good convergence even in scenarios where data is distributed differently, non-independent and identical way between client devices.*

1. Introdução

Os avanços na computação de borda tais como *Edge Computing, Fog Computing* e, de forma mais generalizada, *Multi-access Edge Computing* – MEC [Porambage et al. 2018, Abbas et al. 2018], abriram caminho para o treinamento de modelos de aprendizado de máquina de maneira distribuída, utilizando os recursos dos dispositivos finais (isto é, dispositivos localizados na borda da rede) [Lim et al. 2020]. Uma das técnicas que explora a computação de borda para treinamento de modelos distribuídos é o aprendizado federado (*Federated Learning* – FL) [Lim et al. 2020, Abdulrahman et al. 2021].

O FL permite um treinamento de modelos distribuídos preservando a privacidade dos dados dos usuários. Para isso, os dispositivos (clientes) treinam um modelo compartilhado com seus dados locais. Em seguida, esse modelo é compartilhado com um servidor, o qual é responsável por agregar os modelos recebidos dos clientes para permitir o treinamento colaborativo. Por fim, o modelo é atualizado e compartilhado novamente com os clientes para um novo treinamento. Este processo é chamado de rodada de comunicação e é repetido até que o modelo atinja a convergência. Em outras palavras, o FL permite que dispositivos distribuídos treinem de forma colaborativa um modelo compartilhado, mantendo todos os dados de treinamento no dispositivo e compartilhando apenas o modelo treinado localmente. Dessa forma, é possível mitigar muitos dos riscos e custos sistêmicos de privacidade resultantes de abordagens de aprendizado de máquina tradicionais e centralizadas [Zhang et al. 2020].

Um fator essencial que deve ser considerado em soluções de aprendizado federado é o custo de comunicação, o qual potencialmente introduz um *overhead* adicional para o sistema [Wahab et al. 2021]. Dessa forma, o compartilhamento do modelo entre clientes e servidor deve ocorrer de maneira eficiente. Diversos fatores impactam o *overhead* de comunicação produzido pelo aprendizado federado, incluindo: *(i)* tamanho do modelo compartilhado; *(ii)* quantidade de rodadas de comunicação para alcançar a convergência do modelo; *(iii)* quantidade de clientes participantes por rodada de comunicação; e *(iv)* frequência de compartilhamento. Nesse cenário, uma pergunta de pesquisa ainda em aberto é: como reduzir o *overhead* de comunicação sem degradar a eficiência do modelo treinado?

As principais abordagens para reduzir o *overhead* de comunicação concentram-se na seleção de clientes e na compressão do modelo compartilhado [Wahab et al. 2021]. Entretanto, essas abordagens não são adaptativas, o que pode resultar em soluções ineficientes devido à heterogeneidade de dispositivos e também das mudanças de contexto do ambiente. Por exemplo, *Power-Of-Choice Selection* [Cho et al. 2020] define uma quantidade de clientes fixa para todas as rodadas de comunicação, enquanto as soluções de compressão utilizam as mesmas configurações para todos clientes durante todas rodadas de comunicação.

Nesse contexto, este trabalho apresenta DEEV[1], uma solução para comunicação eficiente em aprendizado federado. DEEV seleciona clientes de forma adaptativa baseado no desempenho do modelo global para acelerar a convergência do modelo enquanto reduz o *overhead* de comunicação e computação no treinamento. Os resultados de experimentos realizados mostram que o DEEV é capaz de reduzir o *overhead* de comunicação em até

[1] `https://github.com/AllanMSouza/DEEV`

60% e em mais de 90% o custo computacional quando comparado com as soluções da literatura.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados, destacando suas vantagens e desvantagens. A Seção 3 descreve a solução proposta, enquanto a Seção 4 apresenta os resultados obtidos pela solução. Por fim, a Seção 5 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Existem desafios em muitos níveis relacionados à comunicação entre clientes e o servidor central em um ambiente de aprendizado federado. Esta seção analisa diferentes abordagens encontradas na literatura para reduzir os custos de comunicação.

A proposta *Federated Averaging* (FedAvg) [McMahan et al. 2017] visa reduzir o número de rodadas de comunicação entre os clientes e o servidor central. No FedAvg, ao invés dos clientes comunicarem atualizações após cada iteração, estes executam várias iterações antes de enviar seus gradientes locais para o servidor. Com base em experimentos realizados em várias arquiteturas de redes neurais convolucionais e recorrentes, os autores mostraram que a comunicação do gradiente local pode ser atrasada por até 100 iterações sem afetar significativamente o treinamento do modelo global, sob a condição de que os dados dos clientes sejam independentes e identicamente distribuídos (IID).

A abordagem *Federated Periodic Averaging and Quantization* (FedPAQ) [Reisizadeh et al. 2020] aborda os desafios de comunicação e escalabilidade no aprendizado federado. Um dos principais recursos do FedPAQ é a média periódica, em que os modelos são atualizados localmente nos dispositivos e a média é calculada apenas periodicamente no servidor. Além disso, o FedPAQ usa um método de quantização para o envio de mensagens do servidor para os clientes com as atualizações locais. No entanto, como no FedAvg, o FedPAQ funciona bem apenas quando os dados nos clientes são distribuídos de forma independente e idêntica.

Outra abordagem para reduzir os custos de comunicação é diminuir o tamanho das mensagens trocadas entre os clientes e o servidor central. Métodos de compressão, como a quantização [Lin et al. 2018, Amiri et al. 2020, Bernstein et al. 2018] e a esparsificação [Shah and Lau 2021, Sattler et al. 2020, Ström 2015], são integrados aos algoritmos de treinamento. Embora reduzam os custos gerais de comunicação, tais métodos aumentam o tempo de convergência e também degradam o desempenho do modelo [Shahid et al. 2021].

Os métodos de esparsificação restringem as mudanças a um pequeno subconjunto dos parâmetros do modelo global [Shahid et al. 2021]. Ström apresenta uma abordagem em que apenas os gradientes dos pesos com magnitude maior que um determinado limite predefinido são enviados ao servidor [Ström 2015]. Shah & Lau propõem uma estrutura para desenvolver modelos compactados onde cada cliente treina um modelo esparso localmente e depois o comunica ao servidor [Shah and Lau 2021]. Como os dados não são IID entre todos os clientes no ambiente FL, o modelo atualizado no servidor pode não ser mais esparso após a média federada. Para recuperar a perda de esparsidade, o servidor central realiza uma reconstrução esparsa do modelo global. Há duas vantagens nessa operação. Primeira, diminui o volume de comunicação *downstream* do servidor para os

clientes. Segunda, permite que os clientes treinem localmente com um modelo já esparso em vez de um denso.

A quantização é outra abordagem de compressão que reduz os gradientes a valores de baixa precisão para diminuir a comunicação cliente-servidor [Lin et al. 2018, Shahid et al. 2021]. Amiri *et al.* mostraram que a aplicação de técnicas de quantização ao modelo global pode ajudar a aumentar a eficiência da comunicação [Amiri et al. 2020]. O servidor central usa o algoritmo *Lossy FL* (LFL) para transmitir o modelo global quantizado para os clientes. As atualizações locais dos clientes e enviadas ao servidor central são quantizadas. Seus experimentos mostram que o LFL proporciona uma redução significativa na carga de comunicação, sem nenhuma lacuna visível no desempenho do cenário totalmente sem perdas. Bernstein *et al.* criaram o SignSGD que apenas transmite o sinal de cada gradiente estocástico de mini-lote para aliviar o problema de gargalo de comunicação de gradientes [Bernstein et al. 2018]. No entanto, o SignSGD não apresenta um bom desempenho em cenários com dados não IID.

Outra abordagem visa reduzir o número de clientes que treinam o modelo em cada rodada de comunicação. Segundo [Cho et al. 2020] vários trabalhos da literatura consideram apenas a participação de uma fração de clientes, onde eles são selecionados aleatoriamente ou baseados na proporção de seus conjunto de dados. Cho *et al.* propõem *Power-Of-Choice* (POC), uma estrutura de seleção de clientes eficiente em computação e comunicação que direciona a seleção de clientes para clientes com maior perda local. Essa seleção permite uma convergência de erros mais rápida e aumenta a eficiência da comunicação em ambientes heterogêneos. Cho *et al.* afirmam que o POC foi projetado para incorrer em sobrecarga mínima de comunicação e computação, aumentando a eficiência de recursos em um ambiente FL. Entretanto, o POC define o número de clientes (isto é, k clientes) que serão selecionados de maneira fixa, ou seja, esse número k deve ser definido previamente e não é ajustado de acordo com o desempenho do modelo, podendo fornecer seleções redundantes e até mesmos desnecessárias mesmo após a convergência do modelo.

Motivado pelas limitações das soluções da literatura, propomos DEEV uma abordagem para seleção de clientes adaptativa, a qual reduz o custo de comunicação e computação para treinamento do modelo em ambientes de aprendizado federado. Além disso, DEEV não define um parâmetro fixo de clientes a serem selecionados e também ajusta a quantidade de clientes selecionados de acordo com o desempenho do modelo.

3. DEEV: Proposta para Seleção de Clientes Adaptativa

DEEV foi proposto para reduzir o *overhead* de comunicação e processamento gerado por soluções de aprendizado federado, sem degradar a eficiência do modelo de aprendizado de máquina. Para isso, DEEV faz uma seleção adaptativa de clientes de acordo com o desempenho do modelo para decidir quais dispositivos devem realizar o treinamento na próxima rodada de comunicação e, consequentemente, permitir um menor *overhead* e um maior desempenho.

3.1. Definição do Problema

Considere um ambiente de aprendizado federado com um total de N clientes, onde cada cliente $n \in N$ possui um *dataset* local D_n com $|D_n|$ amostras. Cada cliente treina um

modelo W localmente e compartilha seus pesos aprendidos W_n com um servidor central, o qual é responsável por agregar os modelos recebidos e encontrar os melhores parâmetros W^* que minimize:

$$F(W) = \frac{1}{\sum_{n \in N} |D_n|} \sum_{n \in N} \sum_{d_n \in D_n} f(W_n, d_n) = \sum_{n \in N} p_n F_n(W_n), \tag{1}$$

onde $f(W_n, d_n)$ é a função de perda para a amostra $d_n \in D_n$ e os pesos W_n aprendidos pelo cliente n, respectivamente. O termo $p_n = \frac{|D_n|}{\sum_{n \in N} |D_n|}$ é a fração de dados do cliente n, e $F_n(W_n) = \frac{1}{|D_n|} \sum_{d \in D_n} f(W_n, d_n)$ é a função objetivo local do cliente n.

O FedAvg [McMahan et al. 2017] permite solucionar a Equação (1), o qual divide o treinamento em rodadas de comunicação e seleciona um conjunto de clientes aleatoriamente para participar do treinamento. Cada cliente treina o modelo em seu conjunto de dados locais com τ iterações (*epochs*) usando o algoritmo de gradiente descendente estocástico (*Stochastic Gradient Descent* - SGD) [Haykin 1998] e envia o modelo treinado de volta para o servidor. Dessa forma, o servidor atualiza o modelo global e compartilha novamente com um novo conjunto de clientes.

Formalmente, o conjunto de $0 < n \leq |N|$ clientes selecionados pelo servidor é representado por $\mathcal{S}^{(t)}$. Dessa forma, os clientes executam τ iterações locais e o conjunto de clientes se mantém constante em cada iteração. Assim, para $(t+1) \bmod \tau \neq 0$, temos $\mathcal{S}^{(t+1)} = \mathcal{S}^{(t+2)} = \ldots = \mathcal{S}^{(t+\tau)}$. Portanto, a atualização do FedAvg ocorre da seguinte forma:

$$W_n^{(t+1)} = \begin{cases} W_n^{(t)} - \eta_t g_n(W_n^{(t)}, d_n) & \text{para } (t+1) \bmod \tau \neq 0 \\ \frac{1}{|\mathcal{S}^{(t)}|} \sum_{m \in \mathcal{S}^{(t)}} (W_m^{(t)} - \eta_t g_m(W_m^{(t)}, d_m)) \triangleq \overline{W}^{(t+1)} & \text{Caso contrário} \end{cases} \tag{2}$$

onde $W_n^{(t+1)}$ representa o modelo local do cliente n na iteração t, η_t é a taxa de aprendizado e $g_n(W_n^{(t)}, d_n)$ representa o SGD sobre a fração de dados d_n do cliente n. Além disso, $\overline{W}^{(t+1)}$ define o modelo global no servidor. Embora $\overline{W}^{(t)}$ só seja atualizado a cada τ iterações, por questões de convergência, consideramos uma sequência de $\overline{W}^{(t)}$ que é atualizado a cada iteração, dada por:

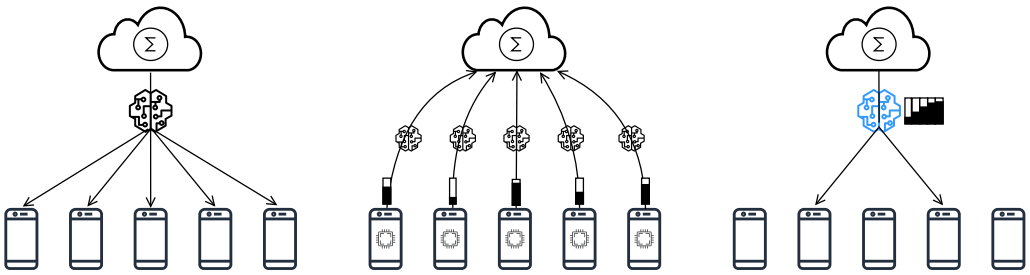
$$\overline{W}^{(t+1)} = \overline{W}^{(t)} - \eta_t \overline{g}^{(t)} = -\frac{\eta_t}{m} \sum_{n \in \mathcal{S}^{(t)}} g_n(W_n^{(t)}, d_n^{(t)}) \tag{3}$$

onde $\overline{g}^{(t)}$ representa a média dos modelos de $w_n^{(t)}$. Mais detalhes sobre a convergência dos modelos com seleção parcial de clientes é apresentada detalhadamente em [Cho et al. 2020].

3.2. Seleção de Clientes baseada em Desempenho

O treinamento de modelos por clientes com desempenho limitado permite alcançar uma convergência mais rápida e um melhor desempenho médio das soluções baseadas em aprendizado federado [Cho et al. 2020]. Assim, clientes que já possuem um bom desempenho *esperam* para que clientes com desempenho inferior treinem seus modelos por mais épocas e, consequentemente, seus desempenhos se aproximem dos demais. Em outras palavras, tal abordagem permite uma convergência mais rápida em direção à média global.

Para isso, além do compartilhamento do modelo, cada cliente deve compartilhar com o servidor uma métrica para que seja possível mensurar o desempenho do modelo em seus dados locais como, por exemplo, a *loss* do treinamento ou até mesmo acurácia. Assim, o servidor tem conhecimento do desempenho de cada cliente e pode aplicar políticas para realizar a seleção com base no seu desempenho. Tais abordagens foram utilizadas para desenvolver DEEV.



(a) Passo 1 (b) Passo 2 (c) Passo 3

Figura 1. Seleção de clientes baseada em desempenho: Passo 1 (a) o servidor envia o modelo para os clientes; no Passo 2 (b) os clientes treinam o modelo com seus dados locais e enviam de volta o modelo treinado para o servidor junto com um métrica de desempenho; no Passo 3 (c), o servidor classifica os clientes de acordo com o desempenho e seleciona os clientes com desempenho inferior a média das acurácias recebidas para treinar o modelo na próxima rodada de comunicação.

A Figura 1 descreve o processo de seleção de clientes aplicado pelo DEEV. Como pode ser visto na Figura 1(a), inicialmente o servidor envia o modelo para todos os clientes. Em seguida, os clientes realizam o treinamento do modelo com seus dados locais e compartilham o modelo com o servidor e também enviam a acurácia obtida após o treinamento, descrito na Figura 1(b). Assim, na Figura 1(c), o servidor recebe os modelos dos clientes, realiza a agregação fazendo uma média ponderada dos modelos e também ordena de maneira crescente o desempenho dos clientes para permitir a seleção de clientes. Por fim, baseado no desempenho, DEEV seleciona os clientes que estão abaixo da acurácia média dos clientes. No exemplo apresentado na Figura 1, é possível identificar apenas dois clientes com desempenho inferior à acurácia média, os quais são selecionados para a próxima rodada de comunicação.

A seleção de clientes implantada pelo DEEV reduz o *overhead* de comunicação introduzido pelo aprendizado federado, pois menos clientes compartilham modelos a cada rodada, enquanto permite uma convergência mais rápida. Além disso, é importante destacar que a seleção implementada pelo DEEV não necessita da definição da quantidade de clientes a priori, pois a mesma é ajustada dinamicamente a cada rodada baseada no desempenho médio dos clientes.

3.3. Seleção Adaptativa

Ajustar a seleção de clientes de acordo com o desempenho dos clientes é essencial para reduzir a utilização de recursos de comunicação e processamento, e também para utilizá-los mais eficientemente. Assim, utilizando uma quantidade fixa de clientes (ou seja, k) a cada rodada, mesmo após a convergência do modelo, os k clientes ainda continuam

sendo selecionados. Outro problema é que clientes que já possuem um bom desempenho e não contribuem para melhorar a convergência do modelo podem ser incluídos na seleção apenas para completar a quantidade definida de k clientes. Isso aumenta o *overhead* de comunicação e processamento da solução.

DEEV implementa uma redução gradual de clientes para que menos clientes sejam selecionados a cada rodada de acordo com o desempenho do modelo. Consequentemente, reduzindo ainda mais a utilização de recursos de comunicação e também de processamento. A redução gradual do número de clientes é implementada por uma função de decaimento dada por:

$$C = \lceil |\mathcal{S}^{(t)}| \cdot (1 - decay)^t \rceil \tag{4}$$

onde C é a quantidade de clientes a serem selecionados, $\mathcal{S}^{(t)}$ representa o conjunto de clientes previamente selecionados pelo DEEV na rodada t e *decay* é um parâmetro para definir quão gradual será o decaimento na quantidade de clientes. A Figura 2 ilustra o decaimento gradual de clientes de acordo com o valor do parâmetro *decay*. É importante destacar que esse decaimento é realizado após a definição do conjunto de clientes adaptativo implementado pelo DEEV. Como pode ser visto, diferentes valores fornecem diferentes decaimentos, os quais devem ser ajustados para permitir uma melhor relação custo-benefício entre desempenho e *overhead*.

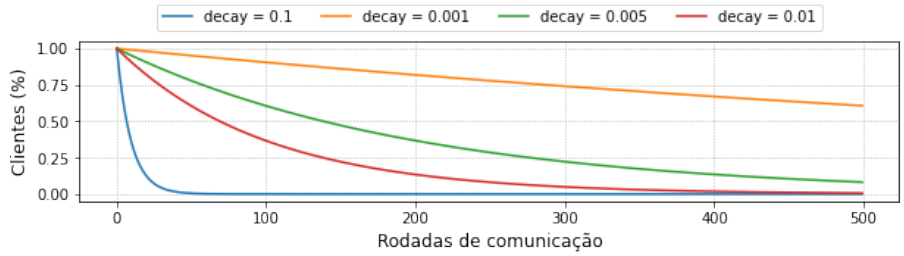


Figura 2. Exemplo decaimento gradual de clientes

Com a quantidade de clientes calculada após a definição do conjunto de clientes que possuem desempenho menor que a média e também após aplicar a função de decaimento, DEEV seleciona os clientes da seguinte forma:

$$n_1, n_2, n_3, \ldots, n_m \text{ para } \forall m \leq C \leq |N|, \tag{5}$$

ou seja, após a ordenação do desempenho dos clientes DEEV seleciona os C primeiros clientes com desempenho inferior à média.

3.4. O Algoritmo DEEV

O Algoritmo 1 descreve o processo completo implementado pelo DEEV, incluindo o compartilhamento do modelo entre clientes e servidor, seleção de clientes, e função de decaimento gradual para permitir um melhor uso da rede de comunicação e também reduzir o *overhead* de processamento.

Como pode ser visto, DEEV inicia um modelo global W com pesos aleatório e define um conjunto de clientes para iniciar o treinamento. Esse conjunto é selecionado baseado em uma porcentagem $p \in [0, 1]$ e definido em $\mathcal{S}^{(t)}$ (Linhas 1-3). Portanto, para

Algorithm 1 DEEV

Servidor executa:

1: $W \leftarrow InicializacaoAleatoria()$ ▷ *inicializa o modelo*
2: $m \leftarrow \max(N \cdot p, 1)$ ▷ *seleciona um conjunto de clientes aleatório* $p \in [0, 1]$
3: $\mathcal{S}^{(t)} \leftarrow SelecionaClientes(N, m)$
4: **for** cada rodada de comunicação $t \in \{1, 2, \ldots\}$ **do**
5: **for** $n \in \mathcal{S}^{(t)}$ **do** ▷ *em paralelo*
6: $W_n^{(t+1)} \leftarrow enviaRequisicaoTreinamentoLocal(n, W^{(t)})$
7: **end for**
8: $W_n^{(t+1)} \leftarrow \sum_{n \in \mathcal{S}^{(t)}} p_n W^{(t+1)}$ ▷ *agrega os gradiente recebidos*
9: $W^{(t+1)} \leftarrow W^{(t-1)} - \Delta^{(t)}$ ▷ *atualiza modelo global*
10: $\mathcal{S}^{(t)} \leftarrow SelecionaClientes(N, W^{(t+1)})$ ▷ *seleciona os clientes com maior loss para próxima rodada de treinamento*
11: **end for**

12: **TreinamentoLocal(n, W):** ▷ *quando o cliente* $n \in \mathcal{S}^{(t)} \mid \mathcal{S}^{(t)} \subseteq N$ *recebe uma requisição de treinamento*
13: **for** cada época $\in 1, 2, \ldots, \tau$ **do**
14: $W_n^{(t)} \leftarrow f(W, D_n)$ ▷ *treina modelo nos dados locais*
15: **end for**
16: $acc_n \leftarrow acuracia(W_n^{(t)}, D_n)$ ▷ *calcula acurácia do modelo nos dados locais do cliente* n
17: $enviaParaOServidor(W_n^{(t)}, n, |D_n|, acc_n)$ ▷ *compartilha modelo treinado com o servidor*

18: **SelecionaClientes(**N**,** W**):** ▷ *Modelos recebidos dos clientes*
19: $acc_{media} \leftarrow \frac{1}{N} \sum_{n \in N} acc_n$
20: **for** $n \in N$ **do** ▷ *configura próxima rodada de treinamento*
21: **if** $acc_n \leq acc_{media}$ **then** ▷ *checa se o desempenho do cliente* n *foi pior que a média*
22: $\mathcal{S}^{(t)} \cup n$ ▷ *seleciona cliente* n *para treinar o modelo na próxima rodada*
23: **end if**
24: $C \leftarrow \lceil |\mathcal{S}^{(t)}| * (1 - decay)^t \rceil$ ▷ *Define quantos clientes serão selecionados baseado no fator de decaimento*
25: $\mathcal{S}^{(t)} \leftarrow \mathcal{S}^{(t)}[0 : C]$ ▷ *seleciona apenas* C *clientes de* $\mathcal{S}^{(t)}$
return $\mathcal{S}^{(t)}$

cada cliente selecionado (isto é, $n \in \mathcal{S}^{(t)}$) DEEV envia uma requisição de treinamento para que esse cliente realize o treinamento local em paralelo (Linhas 4-7). Após receber o modelo dos clientes selecionados, DEEV realiza a agregação dos modelos usando uma média ponderada pela tamanho do *dataset* local de cada cliente e atualiza o modelo (Linhas 8 e 9). Por fim, um novo conjunto de clientes é selecionado baseado no desempenho de cada um e também aplicando a função de decaimento gradual.

Para o treinamento local de cada cliente é executado o algoritmo SGD por τ épocas. Assim, para cada cliente um novo modelo $W_n^{(t)}$ é gerado após o treinamento. Esse novo modelo é enviado para o servidor para que seja agregado (Linhas 12-17 no Algoritmo 1). Por outro lado, para a seleção de clientes, DEEV primeiramente calcula a média das acurácias dos modelos acc_{media} (Linha 19). Dessa forma, para cada cliente com acurácia menor que acurácia DEEV adiciona no conjunto de clientes que serão selecionados $\mathcal{S}^{(t)}$ (Linhas 20-23). Por fim, DEEV aplica a função de decaimento gradual para reduzir a quantidade de clientes selecionados previamente de acordo com $C = \lceil |\mathcal{S}^{(t)}| \cdot (1 - decay)^t \rceil$. Portanto, após definir a quantidade de clientes que serão selecionados do conjunto $\mathcal{S}^{(t)}$ DEEV seleciona os C primeiros clientes do conjunto $\mathcal{S}^{(t)}$

(Linhas 24 e 25).

4. Análise de Desempenho

Esta seção apresenta a avaliação do DEEV em relação aos trabalhos da literatura FedAvg [McMahan et al. 2017], POC [Cho et al. 2020] e uma variação do DEEV sem a função de decaimento gradual, analisando o desempenho do modelo, a seleção de clientes e também *overhead* de comunicação e processamento. A Subseção 4.1 descreve o ambiente utilizado para o desenvolvimento e avaliação das soluções de aprendizado federado. A Subseção 4.2 descreve o *dataset*, aplicação e o modelo utilizado, enquanto a Subseção 4.3 apresenta as métricas utilizadas. Por fim, a Subseção 4.4 apresenta os resultados obtidos.

4.1. Ambiente de Avaliação

Um ambiente baseado em contêineres (ou seja, *dockerizado*) foi criado para a avaliação, onde a ideia principal é dividir as aplicações servidor e cliente em diferentes máquinas virtuais. Para isso, usamos um *Docker Swarm* para criar várias máquinas virtuais (VMs), que são organizadas como VM *manager* ou VMs *worker*. Além disso, cada aplicativo ou seja, servidor e cliente, é implantado nas VMs como contêineres. Cada tipo de VM é responsável por executar um tipo específico de aplicação. Assim, as VMs *manager* executam a aplicação do servidor, enquanto as VMs *worker* executam uma ou mais aplicações cliente (ou seja, contêineres). A Figura 3 ilustra o ambiente *dockerizado* criado.

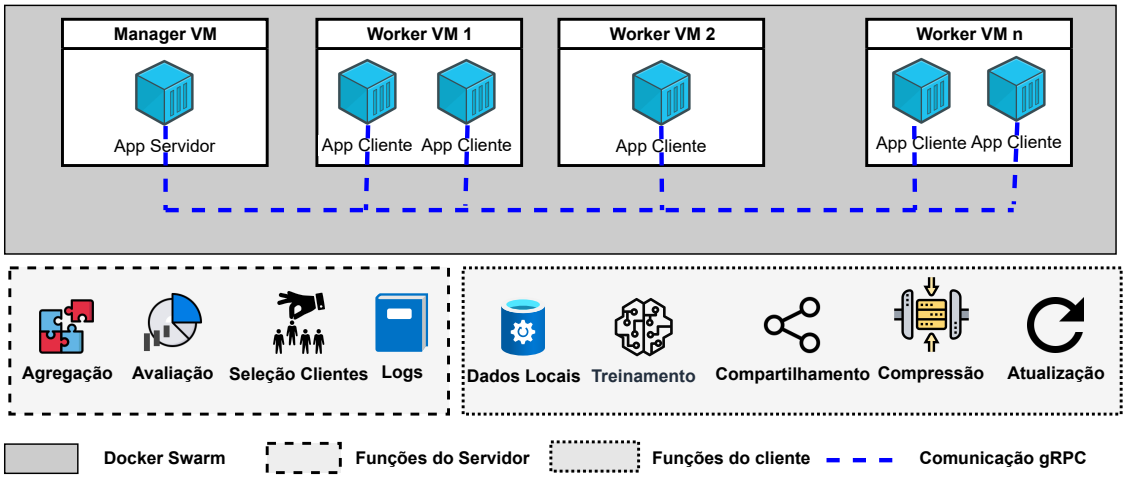


Figura 3. Ambiente *dockerizado* descrevendo a organização das VMs e as tarefas executadas pelo servidor e clientes no processo de aprendizado federado

Como pode ser visto, os contêineres clientes são responsáveis por armazenar os dados locais, treinar o modelo localmente com seus dados, compactar o modelo para melhorar a eficiência da comunicação, compartilhar o modelo treinado com o servidor para proporcionar um aprendizado colaborativo e atualizar o modelo após o servidor realizar a agregação. A comunicação entre os clientes e o servidor é feita por meio de gRPC utilizando o *framework* Flower [Beutel et al. 2020]. Por outro lado, a aplicação servidor é responsável por agregar os modelos recebidos dos clientes, avaliar o desempenho geral do modelo, selecionar um conjunto de clientes para realizar o treinamento na próxima

rodada de comunicação e, finalmente, fornecer *logs* de comunicação, uso de recursos, e métricas de desempenho. Esses *logs* são obtidos usando a API do mecanismo *docker*.

O ambiente é executado a partir de um arquivo *docker-compose*, no qual diferentes parâmetros podem ser passados através de cada contêiner usando variáveis de ambiente, incluindo configuração de rede neural, abordagem de agregação, método de compressão, alocação de recursos (ou seja, limites de CPU e RAM). Vale realçar que diferentes tecnologias de comunicação podem ser usadas conforme definição do atributo de rede do *docker* no arquivo *docker-compose*. Além disso, clientes heterogêneos podem ser definidos personalizando as limitações de CPU e RAM para cada um deles.

4.2. Base de Dados, Aplicação e Modelo

Para avaliar o desempenho dos algoritmos de aprendizado federado, consideramos uma aplicação de reconhecimento de atividades humanas (*Human Activity Recognition – HAR*) utilizando dados de sensores coletados por *smartphones*. Assim, foi utilizada a base de dados MotionSense [Malekzadeh et al. 2019], a qual inclui dados de séries temporais gerados por sensores de acelerômetro e giroscópio (altitude, gravidade, aceleração do usuário e taxa de rotação). Os dados foram coletados com um *iPhone 6s* guardado no bolso frontal de cada participante usando o *SensingKit* que coleta informações do *framework Core Motion* em dispositivos iOS. Todos os dados foram coletados em uma taxa de amostragem de 50 Hz. Um total de 24 participantes, com variações de sexo, idade, peso e altura, realizaram 6 atividades em 15 tentativas no mesmo ambiente e condições: descer escadas, subir escadas, andar, correr, sentar e ficar em pé.

Um modelo adequado para o problema de HAR é o *MultiLayer Perceptron* (MLP) [Haykin 1998] composto por três camadas ocultas, com 256 unidades por camada oculta [Vaizman et al. 2017]. Para o ajuste do MLP, foram utilizados os seguintes parâmetros: o *Stochastic Gradient Descent* (SGD) como método para atualizar os pesos do modelo MLP e o *Sparse Categorical Crossentropy* como função de perda. Queremos otimizar a pontuação de precisão como uma tarefa de classificação. Treinar o modelo no conjunto de treinamento em uma configuração centralizada permitiu que o MLP atingisse uma acurácia de aproximadamente 98% no conjunto de teste. Portanto, o objetivo é treinar o modelo global (MLP) nos dados locais dos clientes, avaliá-lo de forma distribuída e alcançar uma pontuação de precisão o mais próximo possível dos resultados do ambiente centralizado.

4.3. Métricas

Para analisar o desempenho do modelo e também o *overhead* de processamento e comunicação das soluções, as seguintes métricas foram utilizadas:

- **Acurácia do modelo distribuído**: mede a convergência da solução. Assim, a acurácia é calculada de forma distribuída a cada rodada de comunicação. Quanto menos rodadas forem necessárias para atingir a acurácia alvo, mais rápida será a convergência da solução.
- **Seleção relativa de clientes**: mede a redução ou aumento na seleção de clientes do DEEV em relação às soluções da literatura. Valores positivos indicam uma redução na quantidade de clientes da rodada em questão, enquanto valores negativos indicam um aumento na quantidade de clientes.

- **Tempo por rodada de comunicação**: indica o tempo de cada rodada de comunicação para realizar o treinamento do modelo.
- **Bytes transmitidos**: mede a sobrecarga na rede produzida pela solução. É a quantidade de bytes transmitidos por cada solução considerando o tamanho do modelo e o número de clientes selecionados para treinar o modelo localmente.
- **Processamento**: mensura o custo computacional da solução em *Tera Flops* (TFLOPS) para realizar o treinamento do modelo. Os FLOPS definem o custo de processamento de cada neurônio tanto para realizar o *feedforward* e também o *backpropagation*.

4.4. Resultados

Uma análise exploratória foi realizada para definir os melhores hiper-parâmetros para as soluções da literatura. Portanto, para a solução POC, a qual precisa definir a quantidade k de clientes selecionados por rodada, para $k = 12$ tivemos o melhor custo-benefício entre desempenho e *overhead* de comunicação. Por outro lado, definimos a solução FedAvg como *baseline* e, assim, definimos que 100% dos clientes sempre serão selecionados a cada rodada. Por fim, a solução FedSecPer representa DEEV sem o decaimento gradual de clientes.

A Figura 4 apresenta os resultados de acurácia em cada rodada de comunicação. Como pode ser visto, as soluções que realizam a seleção de clientes permitem uma convergência mais rápida do modelo, pois aumenta o desempenho dos clientes com eficiência limitada fazendo com que eles progridam em direção da média global, permitindo um melhor desempenho do modelo. Em particular, o FedSecPer, DEEV e POC atingem a convergência com menos de 100 rodadas de comunicação, enquanto o FedAvg atinge a convergência com aproximadamente 140 rodadas, o que indica um aumento de mais de 90% no *overhead* de comunicação e processamento do FedAvg em relação às outras soluções.

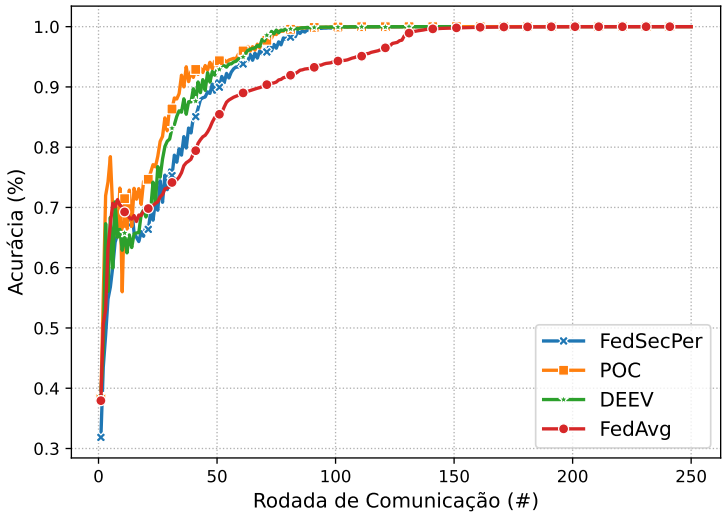
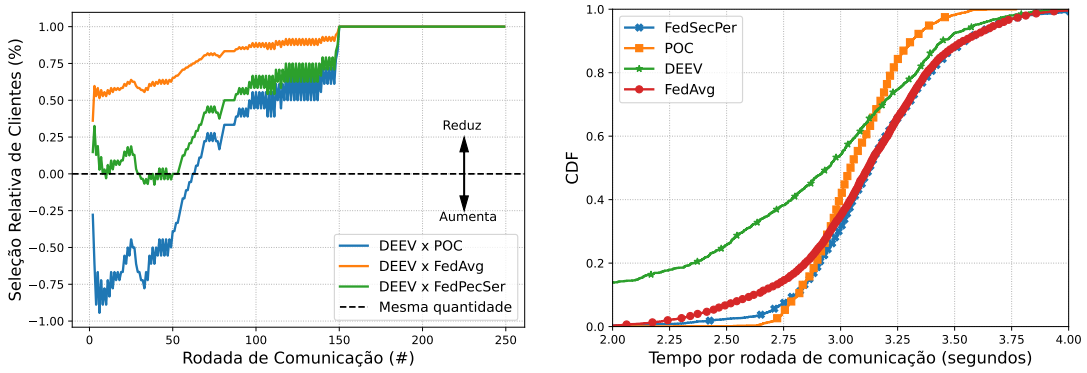


Figura 4. Acurácia por rodada.

A Figura 5(a) descreve os resultados da seleção relativa de clientes do DEEV em relação às outras soluções, mostrando como a seleção adaptativa se comporta ao longo

dos rodadas de comunicação. Uma das vantagens do DEEV em relação ao POC e FedAvg é que ele não necessita definir a priori a quantidade de clientes selecionados, pois essa quantidade é definida dinamicamente de acordo com o desempenho do modelo. Assim, a quantidade de clientes selecionados pode aumentar ou diminuir de acordo com a definição do algoritmo. Quando analisamos a seleção de clientes do DEEV em comparação com o FedAvg, observamos uma redução média de 75%. Por outro lado, quando comparamos DEEV com POC, para as primeiras 50 rodadas, o DEEV aumenta em até 80% a quantidade de clientes selecionados. Entretanto, após a rodada 50, o DEEV começa a reduzir a quantidade de clientes selecionados em relação ao POC. Além disso, é importante destacar que a partir da rodada 150, quando o modelo atinge o desempenho máximo, DEEV para de selecionar os clientes devido ao seu algoritmo de seleção adaptativa, enquanto as outras soluções continuam a selecionar a quantidade definida de clientes. Isso pode ser observado na Figura 5(a), onde a partir da rodada 150 DEEV apresenta uma redução de 100% na seleção de clientes em relação a todas as outras soluções.



(a) Seleção de clientes

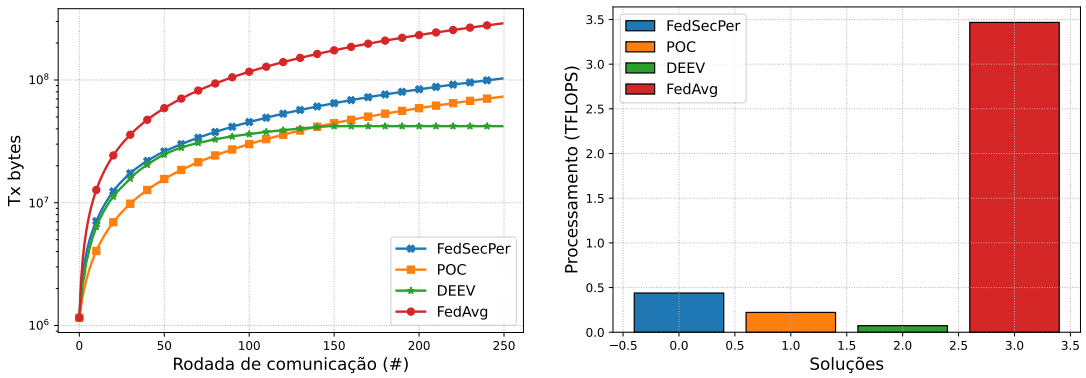
(b) Tempo por rodada

Figura 5. Resultados para seleção relativa de clientes por rodada e tempo gasto por rodada de comunicação.

Analisar o desempenho das soluções em relação à quantidade de rodadas não é suficiente, pois o tempo gasto em cada rodada pode introduzir uma latência indesejada para a solução, degradando a eficiência do sistema. Assim, a Figura 5(b) resume os resultados do tempo gasto por cada rodada em cada solução. DEEV reduz o tempo gasto para realizar o treinamento federado em aproximadamente 70% das rodadas, o que é resultado da seleção eficiente dos clientes. Porém, quando comparado com POC, DEEV apresenta um aumento no tempo gasto para 20% das rodadas. Este comportamento é consequência da maior quantidade de clientes selecionados nas primeiras rodadas. Entretanto, apesar de aumentar o tempo de rodada, tal aumento implica em apenas 0,5 segundo, o que não degrada a eficiência da solução.

O melhor desempenho do DEEV pode ser observado quando analisamos o *overhead* de processamento e comunicação gerado pelas soluções, mostrados nas Figuras 6(a) e 6(b). Como pode ser visto, DEEV gera um menor *overhead* de processamento e comunicação em relação às outras soluções da literatura. Em particular, ele reduz em até 60% o uso de rede e em mais de 90% o custo computacional para o treinamento do modelo. Esses resultados são consequência do algoritmo eficiente de seleção adaptativa de clientes e também do decaimento gradual baseado no desempenho do modelo, permitindo

uma maior escalabilidade para a solução.



(a) Bytes transmitidos (b) Processamento

Figura 6. *Overhead* de comunicação e processamento gerado pelas soluções

Com os resultados apresentados, é possível concluir que: *(i)* a seleção adaptativa de clientes permite uma convergência mais rápida do modelo; *(ii)* o decaimento gradual na seleção de clientes reduz o *overhead* de comunicação e processamento, pois menos clientes são selecionados de acordo com o aumento no desempenho do modelo; *(iii)* é possível reduzir ainda mais o *overhead* de comunicação combinando soluções de compressão de modelos juntamente com a seleção de clientes.

5. Considerações Finais

Este trabalho apresentou DEEV, uma solução de aprendizado federado com seleção adaptativa de clientes baseado no desempenho do modelo distribuído. DEEV também implementa uma função de decaimento gradual para reduzir a quantidade de clientes selecionados a cada rodada de acordo com o desempenho do modelo, levando a uma convergência mais rápida, e também reduzindo o *overhead* de processamento e comunicação.

Para avaliar o desempenho do DEEV, foi utilizada uma base de dados para reconhecimento de atividade humana com dados provenientes de sensores de *smartphones*. Os resultados mostraram que DEEV reduz em até 60% o custo de comunicação e em mais de 90% o custo de processamento para realizar o treinamento do modelo federado em relação às soluções da literatura.

Como trabalhos futuros, serão incorporadas informações sobre os recursos dos dispositivos para permitir uma seleção ciente de contexto de bateria, uso de CPU e memória e até qualidade da rede dos dispositivos. Além disso, algoritmos de compressão de modelos serão desenvolvidos para reduzir ainda mais o *overhead* de comunicação da solução.

6. Agradecimentos

Este projeto foi apoiado pelo Ministério da Ciência, Tecnologia e Inovação, com recursos da Lei nº 8.248, de 23 de outubro de 1991, no âmbito do PPI-Softex, coordenado pela Softex e publicado como Agentes inteligentes para plataformas móveis baseados em tecnologia de Arquitetura Cognitiva (processo 01245.013778/2020-21).

Referências

Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T. (2018). Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465.

Abdulrahman, S., Tout, H., Ould-Slimane, H., Mourad, A., Talhi, C., and Guizani, M. (2021). A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE Internet of Things Journal*, 8(7):5476–5497.

Amiri, M. M., Gündüz, D., Kulkarni, S. R., and Poor, H. V. (2020). Federated learning with quantized global model updates. *ArXiv*, abs/2006.10672.

Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. (2018). Signsgd: compressed optimisation for non-convex problems.

Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Parcollet, T., and Lane, N. D. (2020). Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*.

Cho, Y. J., Wang, J., , and Joshi, G. (2020). Client selection in federated learning: Convergence analysis and power-of-choice selection strategies.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, USA, 2nd edition.

Lim, W. Y. B., Luong, N. C., Hoang, D. T., Jiao, Y., Liang, Y.-C., Yang, Q., Niyato, D., and Miao, C. (2020). Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(3):2031–2063.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, B. (2018). Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*.

Malekzadeh, M., Clegg, R. G., Cavallaro, A., and Haddadi, H. (2019). Mobile sensor data anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI '19, pages 49–58, New York, NY, USA. ACM.

McMahan, B., Moore, E., Ramage, D., Hampson, S., , and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data.

Porambage, P., Okwuibe, J., Liyanage, M., Ylianttila, M., and Taleb, T. (2018). Survey on multi-access edge computing for internet of things realization. *IEEE Communications Surveys & Tutorials*, 20(4):2961–2991.

Reisizadeh, A., Mokhtari, A., Hassani, H., Jadbabaie, A., and Pedarsani, R. (2020). Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization.

Sattler, F., Wiedemann, S., Müller, K.-R., and Samek, W. (2020). Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3400–3413.

Shah, S. M. and Lau, V. K. N. (2021). Model compression for communication efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15.

Shahid, O., Pouriyeh, S., Parizi, R. M., Sheng, Q. Z., Srivastava, G., and Zhao, L. (2021). Communication efficiency in federated learning: Achievements and challenges.

Ström, N. (2015). Scalable distributed dnn training using commodity gpu cloud computing. In *Interspeech 2015*.

Vaizman, Y., Ellis, K., and Lanckriet, G. (2017). Recognizing detailed human context in the wild from smartphones and smartwatches. *IEEE Pervasive Computing*, 16(4):62–74.

Wahab, O. A., Mourad, A., Otrok, H., and Taleb, T. (2021). Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Communications Surveys Tutorials*, 23(2):1342–1397.

Zhang, X., Zhu, X., Wang, J., Yan, H., Chen, H., and Bao, W. (2020). Federated learning with adaptive communication compression under dynamic bandwidth and unreliable networks. *Information Sciences*, 540:242–262.