Privacidade e Comunicação Eficiente em Aprendizado Federado: Uma abordagem utilizando estruturas de dados probabilísticas e políticas de seleção de clientes

Eduardo M. M. Sarmento[1], **Vinícius F. S. Mota**[1], **Rodolfo S. Villaça**[1]

[1] Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Espírito Santo (Ufes)

eduardo.sarmento@edu.ufes.br, vinicius.mota@inf.ufes.br
rodolfo.villaca@ufes.br

Abstract. *To mitigate inference attacks and improve communication efficiency in federated learning, this article proposes a dual approach: i) FedSketch, utilizing sketches to enhance privacy and communication efficiency, applying differential privacy and compression to trained models, and ii) the MetricSelection algorithm, a client selection algorithm based on customized metrics. The proposed solution reduced communication costs, up to 73 times, while maintaining similar accuracy to conventional federated learning with a high level of differential privacy ($\epsilon \approx 10^{-6}$). This represents an effective approach to addressing privacy and cost challenges associated with federated learning.*

Resumo. *Para mitigar ataques de inferência e melhorar a eficiência de comunicação no aprendizado federado, este artigo propõe uma abordagem dupla: i) FedSketch, que utiliza estruturas de dados probabilísticas (sketches) para aumentar a privacidade e eficiência na comunicação, aplicando privacidade diferencial e compactação dos modelos; e ii) MetricSelection, algoritmo de seleção de clientes com base em métricas personalizadas. A solução proposta reduziu o custo da comunicação, em até 73 vezes, mantendo acurácia similar ao aprendizado federado convencional, com altíssimo nível de privacidade diferencial ($\epsilon \approx 10^{-6}$), representando uma abordagem eficaz para enfrentar desafios de privacidade e comunicação associados ao aprendizado federado.*

1. Introdução

O aprendizado federado é uma técnica de treinamento distribuído de redes neurais que tem ganhado destaque nos últimos anos, principalmente quando consideramos aplicações em dispositivos móveis e da Internet das Coisas (IoT) [Zhang et al. 2022]. O aprendizado federado considera que cada dispositivo da federação treina uma rede neural usando apenas os dados locais presentes naquele dispositivo. Posteriormente, cada dispositivo compartilha os parâmetros desta rede neural para um servidor agregação, que os agrega em um único modelo global. O servidor de agregação compartilha o modelo global com os dispositivos, que retreinam seus modelos locais com novos dados. Após várias rodadas de treinamento (*rounds*), obtém-se um processo equivalente ao treinamento com os dados centralizados. Porém, sem compartilhamento de dados durante o processo do treinamento. Os dispositivos que fazem parte do treinamento em cada rodada são escolhidos aleatoriamente ou por meio de algoritmos que consideram métricas de utilidade dos modelos gerados por cada dispositivo [Liu et al. 2022, Souza et al. 2023].

No entanto, dois problemas são conhecidos no aprendizado federado: i) segurança e privacidade dos dados, pois agentes maliciosos que interceptarem os parâmetros das redes neurais, compartilhados com o servidor de agregação, podem, ainda, inferir informações sobre os dados locais de cada treinador e reconstruí-los [Liu et al. 2022]. Outra fragilidade da técnica de aprendizado federado envolve alto custo de transmissão de dados, pois os modelos de redes neurais profundas podem chegar a ter bilhões de parâmetros [Souza et al. 2023].

Uma abordagem para minimizar ataques de inferência aos modelos de redes neurais consiste em aplicar técnicas de privacidade diferencial nos dados antes do treinamento. As técnicas clássicas de privacidade diferencial visam inserir ruído sobre a base de dados original, sem alterar a distribuição dos dados [Dwork 2006]. Contudo, permanece o alto custo de comunicação entre os dispositivos e o servidor de agregação. Visando a compactação de dados com propriedades de privacidade diferencial, as estruturas de dados probabilísticas, conhecidas como *sketches*, são comumente utilizados para contagem de elementos distintos em diversas aplicações de *streaming* de dados massivos, tais como mineração de dados, monitoramento de tráfego de redes, entre outras [Smith et al. 2020]. No contexto de aprendizado federado, técnicas de privacidade diferencial foram utilizadas, por exemplo, para evitar a reconstrução dos dados a partir dos parâmetros dos modelos de redes neurais em aplicações de Internet veicular [Abdel-Basset et al. 2022]. Já os *sketches* têm sido recentemente usados na comunicação eficiente de modelos de redes neurais privados [Li et al. 2019].

Neste sentido, este trabalho propõe: *i*) *FedSketch*, um algoritmo para aumentar a privacidade de dados provendo uma comunicação eficiente por meio do uso de *sketches* e aplicar privacidade diferencial sobre os modelos treinados pelos clientes e, ao mesmo tempo, compactá-los, diminuindo significativamente os custos de comunicação. *ii*) *MetricSelection*, um algoritmo que define políticas de seleção de clientes, para a escolha dos dispositivos que devem realizar e transmitir seus modelos. No *FedSketch*, os modelos são compactados utilizando *sketches*, transmitidos ao servidor de agregação, que por sua vez, agrega os modelos em um modelo global e os envia aos clientes. Estes descompactam, avaliam a utilidade do modelo global e retornam o resultado desta avaliação ao servidor de agregação por meio de métricas de desempenho. O servidor de agregação também é responsável por avaliar essas métricas recebidas dos clientes e definir uma política de quais clientes serão selecionados para uma próxima rodada de treinamento, por meio do algoritmo *MetricSelection*. Foram propostas e avaliadas duas métricas diferentes de seleção de clientes: a primeira usando uma relação similaridade de cosseno entre os *sketches* locais e global, e a segunda usando o próprio desempenho dos modelos locais para a escolha dos treinadores em cada *round*.

Para avaliar os algoritmos propostos, foi implementado uma arquitetura distribuída, na qual as entidades, servidor de agregação e clientes se comunicam por meio de mensagens MQTT. Foram comparados as abordagens com e sem *sketches*, bem como com políticas de seleção de clientes *versus* seleção aleatória, utilizando três *datasets* clássicos da literatura: MNIST, FAMNIST e CIFAR10. Embora *sketches* conhecidamente causem perda de informação durante a compreensão dos dados, este trabalho demonstra que o *FedSketch* reduziu em até 73 vezes o tamanho do modelo, mantendo acurácia similar ao uso clássico de aprendizado federado, e alto nível de privacidade diferencial.

2. Trabalhos Relacionados

A privacidade diferencial [Dwork 2006] é uma técnica usada para garantir a privacidade no compartilhamento público de um conjunto de dados, definida da seguinte maneira: seja M um mecanismo aleatório, no qual $D1$ e $D2$ são as entradas deste mecanismo, que podem diferir entre si em no máximo um elemento. Seja S a saída deste mecanismo:

$$\frac{Pr[M(D1) \in S]}{Pr[M(D2) \in S]} \leq e^{\epsilon} \tag{1}$$

A Equação 1 define que a razão entre as duas probabilidades deve ser menor do que e^{ϵ}, no qual ϵ é um parâmetro de privacidade, chamado *privacy loss* ou *privacy budget*. A definição do valor de ϵ é responsável por um *tradeoff* entre privacidade-utilidade dos dados. Valores baixos de ϵ aumenta a privacidade inserindo mais ruído, enquanto valores maiores insere menos ruído, mas impacta a privacidade dos dados.

Aplicar privacidade diferencial para melhorar a privacidade dos clientes envolvidos no processo de aprendizado federado é uma abordagem promissora. Esta técnica foi primeiro proposta por Wei *et al.* [Wei et al. 2020], onde os autores propõem um algoritmo em que um ruído gaussiano é adicionado tanto aos pesos de cada cliente antes de seu envio ao servidor, quanto aos pesos globais antes de seu envio aos clientes. Husnoo *et al.* [Husnoo et al. 2022] treinam redes neurais para a previsão de consumo de energia residencial, alterando a técnica de maneira a só adicionar o ruído aleatório na transmissão dos pesos enviados pelos clientes ao servidor, com o intuito de melhorar a acurácia atingida e melhorar a privacidade dos usuários.

As estruturas de dados probabilísticas, ou *sketches*, têm sido utilizadas para contagem de elementos em diversas aplicações de *streaming* de dados massivos, tais como mineração de dados, análise de grafos, monitoramento de tráfego, etc [Smith et al. 2020]. A utilização de *sketches* apresenta duas vantagens: i) Redução da cardinalidade dos dados, em alguns casos chegando a poucos kB; e ii) Por consequência dessa redução, os *sketches* também atuam como um mecanismo de privacidade diferencial.

No contexto de aprendizado federado, o tamanho dos vetores de pesos calculados pelas redes neurais pode variar de milhares a bilhões de posições. Neste sentido, em [Konečnỳ et al. 2016], os autores focam em mecanismos para comunicação eficiente dos modelos que cada dispositivo transmite para o servidor durante o processo. Em sua proposta, os autores combinam os conceitos de *subsampling*, no qual apenas uma parte do modelo é local é selecionado, e quantização e cada valor de um vetor de pesos é reduzido para poucos *bits*. Desta forma, os autores mostraram que transmitindo aproximadamente metade dos dados que seriam enviados originalmente, é possível conseguir uma acurácia semelhante. [Li et al. 2019] usam *count sketches* para a compressão dos pesos do modelo de rede neural gerada no treinamento de um dispositivo. Adicionalmente, os autores demonstram que comprimir os pesos em um *count sketch* já é o suficiente para constituir um mecanismo aleatório, devido às colisões que ocorrem durante essa compressão, e sendo assim, satisfazendo a Equação 1, citada anteriormente. [Haddadpour et al. 2020] propõem uma melhoria ao algoritmo proposto por [Li et al. 2019], introduzindo um *learning rate* global e mudando a maneira como os *count sketchs* são descomprimidos com o intuito de melhorar sua convergencia.

A seleção de clientes aleatória no aprendizado federado assume que todos os

clientes envolvidos são iguais, tanto na qualidade de modelos enviados, quanto nos objetivos da federação e na disponibilidade de recursos computacionais. Porém, em muitos cenários realistas esta premissa pode não ser verdadeira. Nishio and Yonetani [Nishio and Yonetani 2019] tentam mitigar este problema propondo um algoritmo de aprendizado federado em que o servidor solicita aos clientes algumas informações de seus recursos computacionais, do tamanho de seu conjunto de dados e estima o tempo necessário para que estes clientes consigam treinar seus modelos. Aqueles que tenham os tempos de treinamento maiores que uma referência são descartados do processo.

Isik-Polat *et al.* [Isik-Polat et al. 2023] propõem um algoritmo em que os pesos de cada cliente são comparados entre si pelo servidor, e clientes que tenham pesos em que pelo menos uma camada seja considerada *outlier* são descartados antes de serem agregados. Já Wang *et al.* [Wang et al. 2022] usa uma medida de qualidade do modelo para selecionar os clientes em cada *round* de treinamento, fixando um limite mínimo de acurácia e escolhendo apenas clientes que atinjam ou superem este limite repetidas vezes durante o treinamento. Souza *et al.* [Souza et al. 2023] propõem um algoritmo em que os clientes são selecionados a partir da acurácia de seus modelos. Na proposta, os clientes com menor acurácia, ou seja, aqueles que precisam de mais tempo de treinamento para aprimorarem a qualidade de seus modelos, são os escolhidos, junto a um mecanismo de redução do no número de clientes participando de cada *round* do treinamento.

Finalmente, a Tabela 1 apresenta um quadro comparativo entre este trabalho e os demais trabalhos citados nesta seção. Como pode-se observar, a proposta apresentada neste artigo é o único que aborda o uso simultâneo de *sketches* para a compactação de pesos, privacidade diferencial e seleção de clientes não aleatória, sendo essa junção de ambas soluções a principal contribuição deste artigo.

Tabela 1. Quadro comparativo de trabalhos relacionados.

Referência	Privacidade Diferencial	Seleção de Clientes não aleatória	Compactação com *Sketches*
[Wei et al. 2020]	✓	x	x
[Husnoo et al. 2022]	✓	x	x
[Smith et al. 2020]	x	x	✓
[Konečnỳ et al. 2016]	x	x	✓
[Li et al. 2019]	✓	x	✓
[Haddadpour et al. 2020]	✓	x	✓
[Nishio and Yonetani 2019]	x	✓	x
[Wang et al. 2022]	✓	✓	x
[Souza et al. 2023]	x	✓	x
Este artigo	✓	✓	✓

3. Solução Proposta

A Figura 1 apresenta uma visão geral da proposta. Na figura, os clientes possuem seus Dados Locais e executam o treinamento de uma rede neural sobre estes dados (1). Os modelos resultantes do treinamento desta rede neural (Modelo Local) são compactados em um *count sketch*, o *Sketch* Local C (2). O dispositivo então avalia se o *sketch* gerado atende os requisitos de privacidade e transmite para o `Servidor de agregação` (3).

Ao receber os *sketches*, o `Servidor de agregação` realiza a média dos valores dos vetores recebidos (4). Em seguida, o servidor envia os *sketches* agregados, C^G para os clientes atualizarem seus modelos (5). Ao receber os *sketches*, cada cliente descompacta o sketch e atualiza seu modelo local (6) e avalia a sua utilidade (7). A utilidade

de um modelo pode ser baseada nas métricas de acurácia que o modelo agrega, na similaridade de cosseno do modelo enviado por um cliente com o modelo global, ou da perda (*loss*) do modelo em relação ao modelo global. Cada cliente envia a métrica definida previamente, no início do processo, de volta para o `Servidor de agregação` (8).

Em seguida, o `Servidor de agregação`, utilizando-se do algoritmo *MetricSelection*, aplica uma política de seleção para escolher os clientes que participarão da próxima rodada de treinamento (9). As políticas de seleção consideradas neste trabalho podem ir desde a seleção de todos os clientes, uma proporção aleatória de clientes, ou algum critério baseado nas métricas de utilidade dos modelos dos clientes. Os clientes selecionados realizam o processo de treinamento novamente (10), reiniciando o processo até um número pré-definido de rodadas.

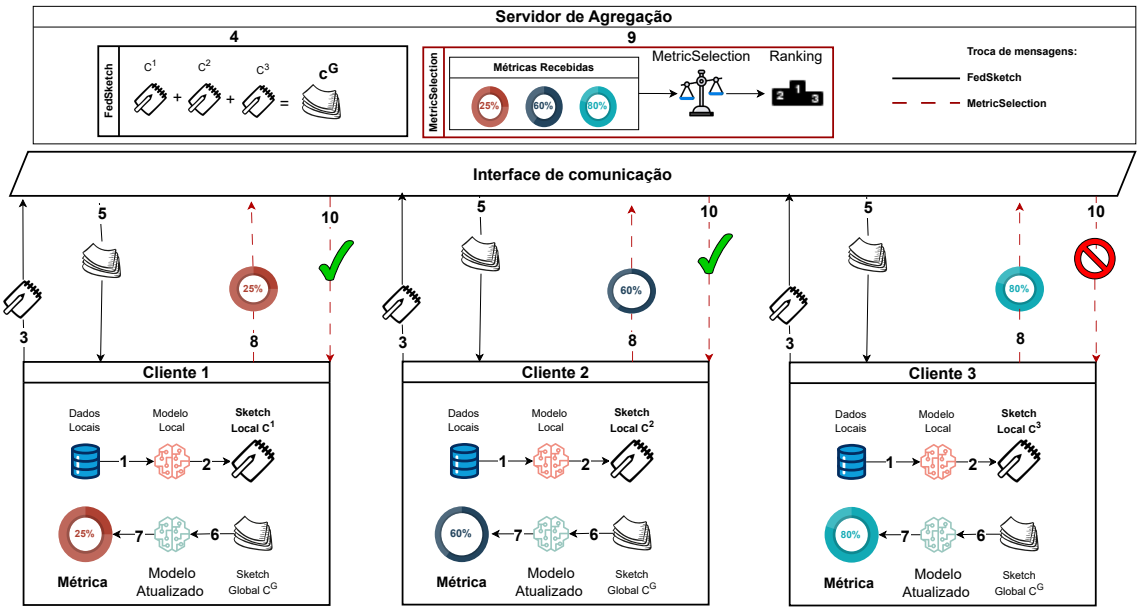

Figura 1. Visão Geral. 1. Treino do `Modelo Local` com `Dados Locais`; 2. Compactação do `Modelo Local` no *`Sketch` `Local`* C; 3. Envio do *`Sketch` `Local`* C ao `Servidor de agregação`; 4. Agregação dos *sketches* locais no *`Sketch` `Global`*, C^G; 5. Retorno do *`Sketch` `Global`* aos clientes; 6. Descompactação do *`Sketch` `Global`* C^G no `Modelo Atualizado`; 7. Avaliação do modelo local com a `Métrica` escolhida; 8. Envio da `Métrica` ao `Servidor de agregação`; 9. Aplicação do *MetricSelection* e do *`Ranking`* às métricas dos clientes; 10. Indicação dos clientes selecionados para o próximo *round*.

3.1. Algoritmo *FedSketch*: Compactação e Descompactação

Após a análise dos trabalhos relacionados, a estrutura de dados *count sketch* foi escolhida para implementação do *FedSketch*. Inicialmente, para facilitar a compreensão do processo é necessário definir quais serão os dados a serem comprimidos nos *sketches*. Neste artigo considera-se que os únicos dados sensíveis transmitidos entre o servidor e os clientes, durante o processo de treinamento, são as atualizações dos pesos dos modelos em cada *round*. Ou seja, no processo, a cada *round* os clientes são responsáveis por salvar seus pesos iniciais, antes do treinamento, denominados w_{old}. Considerando que w_{new} representa os pesos atingidos após um *round* de treinamento, faz-se $w_{delta} = w_{new} - w_{old}$,

onde w_{delta} representa a lista de atualizações dos pesos no round em andamento, que deverá ser transmitida para o servidor. Portanto, a compressão deverá ser efetuada em w_{delta}. Porém, sabendo-se que w_{delta} é uma lista de vetores multidimensionais, para reduzir a complexidade do processo e efetuar a compressão destes dados, torna-se necessário transformar cada vetor multidimensional em um vetor unidimensional. Por fim, estes vetores unidimensionais são concatenados em um único vetor de atualizações local, V, também unidimensional, preservando-se a ordem original em que apareciam em w_{delta}.

O Algoritmo 1 instancia um *count sketch*, definido como uma estrutura de dados probabilística multidimensional, $C^k_{m\times n}$, sendo k o identificador do cliente, composta por m *hashmaps*, sendo que cada *hashmap* possui n posições. Por simplificação de notação, no texto a seguir. $C^k_{m\times n}$ será simplesmente denominado C. Para adicionar um valor x_i, pertencente a um vetor de atualizações V, em um *count sketch* de tamanho $m \times n$, são necessários dois conjuntos de funções de *hashing*, independentes, com m funções cada. O primeiro conjunto contém m funções de *hashing* que mapeiam valores reais em valores inteiros, h_index, sendo que $h_{index} \mapsto 1..n$, e o segundo conjunto, também com m funções de *hashing*, que mapeiam valores reais em dois possíveis valores, 1 ou -1 (h_{sign}). A seguir, para cada *hashmap* presente no *count sketch* calcula-se os *hashes* do índice i do valor $x_i \in V$ usando as funções h_{index} (Linha 4), e h_{sign} (Linha 5), correspondentes ao *hashmap* selecionado. Com o valor retornado por h_{index} tem-se o índice $index$ do valor x_i em seu *hashmap* j ($1 \leq j \leq n$) correspondente no *sketch* $C^k_{m\times n}$. Em seguida, com o valor $sign$, obtido com o uso da função h_{sign}, determina-se se x_i será somado ou subtraído do valor contido em $C[j][index]$, ou seja, na posição $index$ do *hashmap* j do *count sketch* (Linha 6). Após realizar este processo para cada x_i contido no vetor V, ocorrerá a compactação do vetor de atualizações de pesos.

Algoritmo 1 Descreve o processo de compressão do vetor de atualização de pesos, V no *count sketch* $C^k_{m\times n}$. h_{index} e h_{sign} são funções de *hashing*. ϵ_{max} é o nível mínimo de privacidade requerido no processo.

```
1:  Compress(V, C^k_{m×n}, h_index, h_sign, ε_m)
2:  for each x_i ∈ V do
3:      for j ∈ 1, 2...m do
4:          index = h_index[j](i)                          {index < n}
5:          sign = h_sign[j](i)
6:          C[j][index] = C[j][index] + sign * x_i
7:      end for
8:  end for
9:  ε = EpsilonEstimation(V)
10: if ε > ε_max then
11:     AddLaplacianNoise(C)
12: end if
```

Após o processo de compressão, e antes de enviar o *sketch* com as atualizações dos pesos para o servidor, é necessário avaliar o nível de privacidade ϵ do *count sketch*. O cálculo do nível de privacidade obtido com a compactação é realizado utilizando-se das Equações 2 e 3 [Li et al. 2019]. Na Equação 2, β é uma constante positiva calculada usando a Equação 3, σ é o desvio padrão dos valores presentes no vetor de atualizações,

V, v representa a quantidade de elementos x_i presentes do vetor V, α é uma constante, estimada a cada *round*, que deve ser maior que qualquer elemento x_i do vetor V ($\alpha \geq |x_i|, \forall x_i \in V$), m é o número de *hashmaps* presentes no *count sketch* e n é o número de posições de cada *hashmap*.

$$\epsilon = m * ln(1 + \frac{\beta\alpha^2 n(n-1)}{\sigma^2(v-2)}(1 + ln(v-n))) \tag{2}$$

$$\frac{\alpha^2 n(n-1)}{\sigma^2(v-2)}(1 + ln(v-n)) \leq \frac{1}{2} - \frac{1}{\beta} \tag{3}$$

Para avaliar o nível de privacidade obtido, estima-se o ϵ (Linha 9) e o compara com o valor de ϵ_{max}(Linha 10), que representa o nível mínimo de privacidade desejado na compactação - lembrando que quanto maior o valor de ϵ, menor será o nível de privacidade obtido na compactação. O valor de ϵ_{max} é passado como hiperparâmetro para o método. Se $\epsilon > \epsilon_{max}$, um ruído laplaciano deve ser adicionado ao *count sketch* (Linha 11). Conforme observado na Equação 1, quanto mais próximo de zero o valor de ϵ, mais privacidade é oferecida pelo algoritmo, sendo que, segundo o NIST (*National Institute of Standards and Technology*) [Joseph Near 2023], valores menores ou iguais a 1 provém fortes garantias de privacidade para a maioria dos casos reais.

Na sequência, o *count sketch* C^k de cada cliente k é transmitido ao servidor, onde é agregado aos demais *sketches* dos outros clientes. O processo de agregação corresponde à soma, elemento a elemento, dos *count sketches* de todos os clientes divido pelo número de clientes participando do processo de aprendizado. O resultado desta operação é um novo *count sketch* que representa a atualização dos pesos do modelo global (agregação). Este novo *count sketch*, C^G, deve ser transmitido de volta para todos os clientes.

Algoritmo 2 Descreve o processo de descompactação do vetor de atualização de pesos, V no *count sketch* $C^k_{m \times n}$. h_{index} e h_{sign} são funções de *hashing* V_{global} é o vetor de pesos obtido após o processo.

1: $\text{Decompress}(V, C^k_{m \times n}, h_{index}, h_{sign})$
2: $V_{global} = []$
3: **for each** $x_i \in V$ **do**
4: $\quad estimative = []$
5: $\quad$ **for** $j \in 1, 2...m$ **do**
6: $\quad\quad index = h_{index}[j](i)$ $\quad\quad \{index < n\}$
7: $\quad\quad sign = h_{sign}[j](i)$
8: $\quad\quad$ **append** $sign * C[j][index]$ to $estimative$
9: $\quad$ **end for**
10: $\quad x_i = Median(estimative)$
11: $\quad$ **append** x_i to V_{global}
12: **end for**

O processo de descompactação encontra-se sumarizado no Algoritmo 2. Para que cada cliente possa recuperar as atualizações do modelo global, ele deve usar o seu vetor de atualizações V e o *count sketch* global C^G que acabou de receber do servidor. Para cada índice i do vetor V, usando a função h_{index} calcula-se os valores $index$ correspondentes

a i de cada *hash map* j. Em outras palavras $V[i] \mapsto C[j][index]$ (Linha 6). A partir do $index$, recupera-se os valores contidos em C^G e ajusta-se o sinal multiplicando cada valor em $C[j][index]$ pelo resultado do cálculo das funções de *hashing* h_{sign} aplicadas a i (Linha 7). O resultado desta multiplicação deverá ser incluído no vetor $estimative$ (Linha 8). O valor x_i a ser usado para atualização do vetor global, V_{global}, corresponderá à mediana dos valores presentes no vetor $estimative$ (Linha 10). A mediana do vetor $estimative$ é então incluída no vetor V_{global} descomprimido (Linha 11).

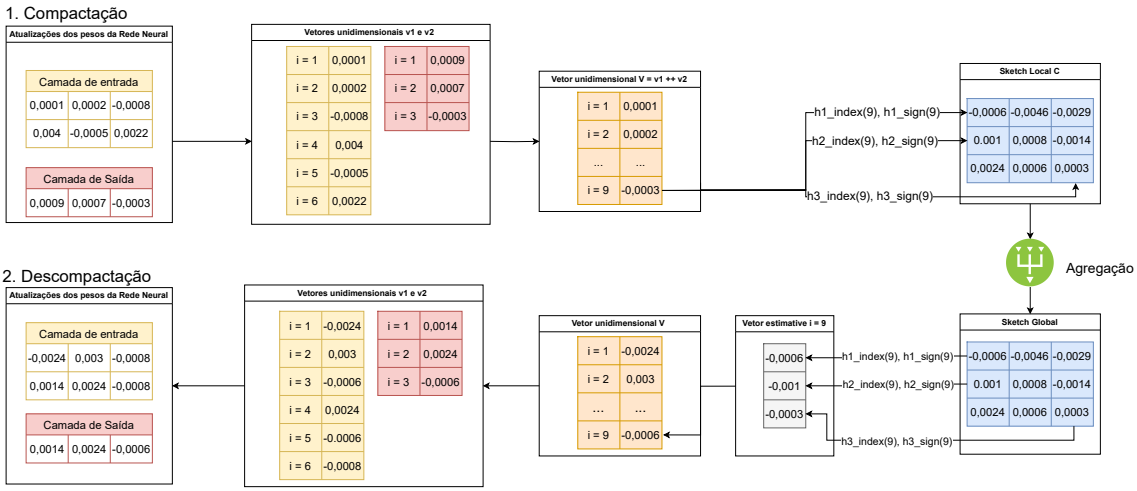

Figura 2. Exemplo de uma execução dos algoritmos de compactação (1) e descompactação (2) das atualizações dos pesos. No exemplo considera-se apenas o peso com índice $i = 9$ no vetor V.

Ao final deste processo, o cliente recuperou o vetor de atualizações global enviado pelo servidor, V_{global}. Porém, este vetor ainda precisa ser transformado para o mesmo formato da lista de vetores de atualizações, w_{delta}, visando transformar o vetor unidimensional V_{global} em um vetor multidimensional com sub-vetores correspondentes aos vetores presentes em w_{delta}. Ao final deste processo tem-se a lista de atualizações globais w_{delta_global}. Para atualizar sua lista de pesos locais em w_{new} os clientes devem então somar a lista de atualizações globais, w_{delta_global} aos pesos locais. Finalmente, termina-se o primeiro *round*, e o processo pode ser reiniciado.

Os algoritmos de compactação e descompactação estão ilustrados e representados na Figura 2. Como exemplo, considere inicialmente uma lista de matrizes contendo as `Atualizações dos pesos da Rede Neural`, encontradas no treinamento de uma rede totalmente conectada e com apenas 1 camada oculta. As matrizes são transformadas nos `Vetores unidimensionais` $v1$ `e` $v2$, posteriormente concatenados $(++)$ para formar o `Vetor unidimensional` $V = v1 ++ v2$. V é então compactado em um `Sketch Local` C, com m=3 *hashmaps* e n=3 posições por *hashmap* usando as famílias de funções de *hashing* h_{index} e h_{sign}, usadas para determinar a posição e o sinal. No exemplo, considere que na posição $i = 9$ do `Vetor unidimensional` V tem-se o valor de peso igual a $-0,0003$. Considere, também, que o resultado de $h1_{index}(9) = 1$, $h2_{index}(9) = 1$ e $h3_{index}(9) = 3$. Em azul, no `Sketch Local` C, os valores correspondentes em $C_{11} = -0,0006$, $C_{21} = 0,001$ e $C_{33} = 0,0003$ correspondem ao resultado da soma $(h_{sign} > 0)$ ou subtração $(h_{sign} < 0)$ de $V[i]$ destes valores aos valores anterior-

mente contidos nas suas respectivas posições no $Sketch$ Local C.

Após o processo de agregação, realizado no servidor, tem-se um $Sketch$ Global que deve ser descomprimido pelo cliente. Para a descompressão, sabe-se previamente o tamanho do vetor V e, por isso, percorrem-se os índices de 1 até o tamanho de V e usam-se as famílias de funções de *hashing* h_{index} e h_{sign} para compor o vetor de estimativa de cada índice, denominado Vetor estimative. No exemplo, ilustra-se somente a descompressão doVetor estimative para o último elemento do Vetor unidimensional V, ou seja, $i = 9$.

Considere, novamente, que o resultado de $h1_{index}(9) = 1$, $h2_{index}(9) = 1$ e $h3_{index}(9) = 3$, e que $h1_{sign}(9) > 0$, $h2_{sign}(9) < 0$ e $h3_{sign}(9) < 0$. Desta forma, $estimative[1] = C_{11} = -0,0006$, $estimative[2] = -1 * C_{21} = -0,001$ e $estimative[3] = -1 * C_{33} = -0,0003$. Em seguida, usa-se a mediana do Vetor estimative como o valor estimado para a posição i do novo Vetor unidimensional V que esta sendo considerado naquele momento da iteração, neste caso, $i = 9$. O Vetor unidimensional V então é subdivido nos Vetores unidimensionais $v1$ e $v2$, pois sabe-se, antecipadamente, onde estes vetores começam e terminam em V. Por último, no exemplo, os Vetores unidimensionais $v1$ e $v2$ são transformados em 2 matrizes, que representam a lista de Atualizações dos pesos da Rede Neural. Desta forma, conclui-se o processo de descompactação. É importante destacar novamente que, conforme ilustrado na figura, quanto menores forem as dimensões dos *sketches* usados no processo, maior será a compactação alcançada.

3.2. Algoritmo *MetricSelection*

O *FedAvg* [McMahan et al. 2016], normalmente utilizado no processo de aprendizado federado, escolhe aleatoriamente um número pré-estabelecido de clientes. Porém, caso os clientes sejam heterogêneos demais, seja na distribuição de probabilidade espacial dos seus dados, seja na disponibilidade de seus recursos computacionais, este esquema de seleção de clientes pode não ser eficiente com relação à convergência do modelo global ou da acurácia final deste modelo [Fu et al. 2022].

O *MetricSelection* está descrito no Algoritmo 3. No algoritmo *MetricSelection*, o servidor inicia o processo de treinamento (Linhas 2-5) selecionando todos os clientes k pertencentes a lista de clientes L. Nos *rounds* subsequentes, cada cliente mede a utilidade de seu modelo usando alguma métrica pré-estabelecida atingida naquele *round*. A métrica que será utilizada no *MetricSelection* é um parâmetro de inicialização. Neste algoritmo, cada cliente envia o resultado da avaliação dessa métrica ao servidor, que monta uma lista ordenada, M, com os valores recebidos de cada cliente $k \in L$. A ordenação pode ser feita de maneira crescente ou decrescente, sendo que esta direção também é um parâmetro de configuração do algoritmo *MetricSelection*.

Em seguida, o servidor calcula a média das métricas de todos os clientes e seleciona ($Choose$) apenas aqueles clientes que tiverem métricas melhores ou iguais à média (Linhas 7-13). A tendência do processo é que, a medida que os *rounds* se passam, os modelos locais se tornam mais similares entre si, e desta forma, naturalmente haverá um decaimento na quantidade de clientes participando do processo, e menos clientes serão escolhidos, reduzindo, assim, o custo da comunicação entre os clientes e o servidor.

Algoritmo 3 Descreve o processo de seleção de clientes, com ordenação crescente das métricas em M no algoritmo *MetricSelection*.

1: *MetricSelection*$(M, L, decrescent)$
2: **if** $round == 1$ **then**
3: **for each** $c_k \in L$ **do**
4: $Choose(c_k, TRUE)$
5: **end for**
6: **else**
7: **for each** $m_k \in M$ **do**
8: **if** $decrescent == TRUE$ **then**
9: **if** $m_k \leq mean(M)$ **then**
10: $Choose(c_k, TRUE)$
11: **else**
12: $Choose(c_k, FALSE)$
13: **end if**
14: **else**
15: **if** $m_k \geq mean(M)$ **then**
16: $Choose(c_k, TRUE)$
17: **else**
18: $Choose(c_k, FALSE)$
19: **end if**
20: **end if**
21: **end for**
22: **end if**

4. Implementação, Avaliação e Resultados

Para implementar o processo de aprendizado federado com uso de *sketches* e políticas de seleção, foi desenvolvida uma arquitetura que permite que os clientes se comuniquem com o servidor de agregação utilizando o modelo *publish-subscribe*, por meio do protocolo *Message Queuing Telemetry Transport* (MQTT). Deste modo, um broker MQTT atua como a interface de comunicação na nuvem entre os clientes e o servidor de agregação.

Os experimentos foram executados em um servidor com 128Gb de RAM e uma CPU Intel® Core™ i9-10900K CPU @ 3.70GHz, e a solução foi implementada [1] usando a linguagem de programação Python e um *broker* EMQX. O aprendizado federado neste artigo foi instanciado com 50 clientes e 1 servidor de agregação.

Foram utilizadas as bases de classificação de imagensMNIST[2], FAMNIST[3] e CIFAR10[4]. Para o conjunto de treinamento local de cada cliente, foram escolhidos de forma aleatória, sem reposição, 500 imagens, no caso dos *datasets* MNIST e FAMNIST, e 1000 imagens, no caso do *dataset* CIFAR10. Para compor o conjunto de teste local de cada cliente, foram escolhidas aleatoriamente, também sem reposição, 250 imagens, no caso dos dois primeiros *datasets*, e 500 imagens, no caso do último *dataset*.

[1] Em caso de aprovação deste artigo, todo o código será disponibilizado publicamente no *github*.
[2] `https://huggingface.co/datasets/mnist`
[3] `https://huggingface.co/datasets/fashion_mnist`
[4] `https://huggingface.co/datasets/cifar10`

4.1. Compactação e Avaliação da Privacidade

Considerando que o treinamento de redes neurais tem um alto custo computacional, a permutação da quantidade de hashmaps e posições por hashmaps para os *sketches* pode se tornar inviável. De fato, [Li et al. 2019] já apresentaram um estudo que mostra que quanto mais compactado menor a acurácia. O desafio está em encontrar valores que representem uma compactação, sem prejudicar a acurácia do modelo. Neste sentido, foram realizados pré-testes visando identificar valores dos parâmetros apropriados por cada modelo. A automatização destes parâmetros baseada no cenário é um trabalho futuro.

Para os datasets MNIST e FAMNIST, cada *count sketch* tem 20 *hashmaps* e 41 posições por *hashmap*, enquanto para o *dataset* CIFAR10, foram configurados 20 *hashmaps* e 915 posições por *hashmap*. Estes valores foram escolhidos para com o intuito de manter bons resultados, apesar da compactação. A taxa de compreensão do *count sketch* pode ser obtido pela razão entre $\frac{P}{m*n}$, sendo P o número de parâmetros da rede, m o número de *hashmaps* e n o número de posições por *hashmap*.

Como arquitetura de rede neural, foi utilizada a rede convolucional LeNet-5 [Lecun et al. 1998], que possui em torno de 60000 parâmetros para imagens em preto e branco e em torno de 549010 parâmetros para imagens coloridas. Desta forma, para os datasets datasets MNIST e FAMNIST, o *count sketch*, que tem tamanho 20 $\times$ 41, comprime 73 vezes o modelo original. Já para o *dataset* CIFAR10, o *sketch*, que tem tamanho 20 $\times$ 915, obteve uma compressão de 30 vezes em relação ao modelo original. Ao analisar o nível de privacidade (ϵ), conforme a Equação 2, obteve-se $\epsilon \approx 10^{-6}$, representando um alto nível de privacidade diferencial ($\epsilon < 1$), eficaz para enfrentar desafios de privacidade e comunicação associados ao aprendizado federado.

4.2. Avaliação do *FedSketch*

Para avaliar a relação custo/ benefício da utilização do algoritmo *FedSketch*, foi comparada sua acurácia com o aprendizado federado tradicional, baseado no algoritmo *FedAvg*. Para isso, ambos os algoritmos foram executados nos três *datasets*, sem nenhum algoritmo de seleção de clientes, isto é, todos os 50 clientes participam do processo de treinamento.

A Figura 3 apresenta representa a acurácia média a cada *round* para cada *dataset*, sendo a área sombreada o desvio padrão da acurácia. Além disso, o resultado nomeado de *all FedAvg* corresponde ao algoritmo sem uso de *sketches* e o resultado nomeado de *all FedSketch* corresponde ao algoritmo com uso de *sketches*.

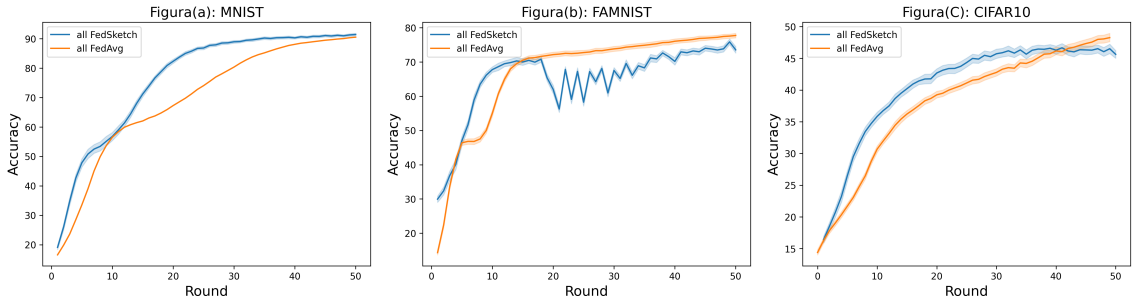

Figura 3. Acurácia média por round dos algoritmos *all FedAvg* e *FedSketch*. Não há nenhuma política de seleção de clientes aplicada nestes resultados. Todos os clientes participam do processo de treinamento.

É possível observar que o desempenho do algoritmo *FedSketch* em comparação com o algoritmo *FedAvg* variou conforme o *dataset* utilizado. Para o *dataset* MNIST o uso de *sketches* levou a um melhor desempenho em todos os *rounds* de treinamento e, por consequência, uma melhor acurácia final, obtendo 91,43% enquanto o *FedAvg* obteve 90,52%. Já para os *datasets* FAMNIST e CIFAR10, o algoritmo *FedSketch* inicialmente se manteve com melhor desempenho, mas ao final foi superado pelo *FedAvg*, obtendo 73,55% e 45,62%, respectivamente, enquanto o *FedAvg* obteve 77,76% e 48,26%, respectivamente. Sendo assim houve uma pequena melhora na acurácia para o *dataset* MNIST com um custo de comunicação 73 vezes menor. Para os *datasets* FAMNIST e CIFAR10 houve uma pequena piora na acurácia final, porém com uma redução do custo de comunicação de 73 e 30 vezes, respectivamente. Este desempenho foi atingido mesmo com os valores ϵ tendo chegado a casa de 10^{-6}, que tem fortes garantias de privacidade.

4.3. Avaliação do *MetricSelection*

O objetivo deste experimento é comparar o algoritmo *MetricSelection*, com o uso de diferentes métricas de comparação, com uma seleção de clientes aleatória (*random*). Em ambos, houve a compactação dos pesos feita a partir do *FedSketch*. Para a seleção de clientes aleatória foi variada a quantidade de clientes em cada *round* entre 50%, 75% e 100% (*all*).As métricas utilizadas para comparação da contribuição, enviadas pelos clientes a cada *round*, para o modelo global foram:

***Accuracy comparison*:** acurácia atingida pelo modelo global, avaliada a cada *round*.
***Sketch comparison*:** similaridade de cosseno entre o *sketch* global e o *sketch* local.

Os resultados estão apresentados na Figura 4. Nela, para facilitar a visualização, estão apresentados os resultados apenas do algoritmo de seleção que obteve os melhores resultados em cada *dataset*. Pode-se notar que, para os *datasets* MNIST e CIFAR10, quanto mais clientes são selecionados de modo aleatório durante o treinamento, mais rápido a convergência do modelo é alcançada. Entretanto, a acurácia final é praticamente o mesmo para todos os algoritmos de seleção de clientes. Para o *dataset* FAMNIST há uma inversão dessa relação, quanto menos clientes foram selecionados para o treinamento, melhor a acurácia final e mais rapidamente se alcançou a convergência dos modelos. Quanto ao resultado dos algoritmos não representados na figura, os resultados são: i) MNIST e CIFAR10, com o *sketch comparison*, 89,59% e 46,52%, respectivamente; e ii) FAMNIST, com o *accuracy comparison*, 69,31%.

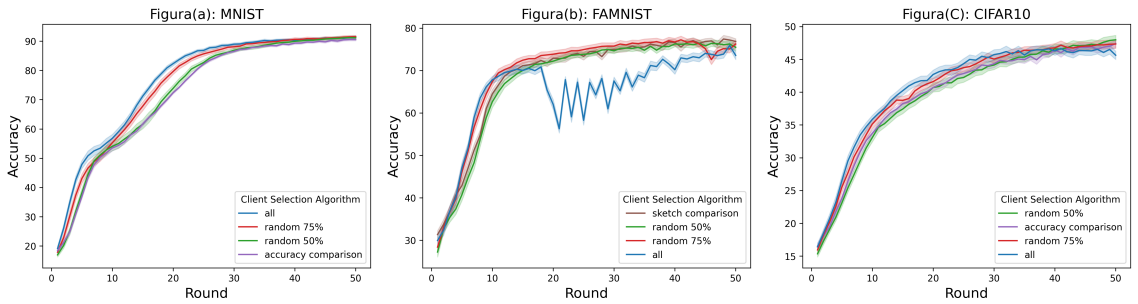

Figura 4. Acurácia média obtida em cada *round* de treinamento para os diferentes algoritmos de seleção de clientes. Variou-se a escolha aleatória entre 50% e 75% e apenas a melhor métrica do *MetricSelection* está representada neste resultado.

Tabela 2. Porcentagem de clientes utilizados no treinamento em cada política de seleção de clientes em cada *dataset* usado nos experimentos.

Algoritmo	MNIST	FAMNIST	CIFAR10
Random 50%	50%	50%	50%
Random 75%	75%	75%	75%
MetricSelection	48%	48%	47%

Entretanto, em todos os casos avaliados, o número médio de clientes selecionados a cada *round* pelo algoritmo *MetricSelection* foi um pouco menor que 50%, conforme resultados apresentados na Tabela 2, que traz a porcentagem de clientes escolhidos (em média) pela seleção aleatória de 50% e 75% dos clientes e pelo algoritmo *MetricSelection*. É possível observar que o *MetricSelection* reduziu o número de clientes selecionados 2 ou 3 p.p. com relação à escolha aleatória de 50% de clientes, obtendo resultados de acurácia similares para todos os *datasets*.

A política de seleção de clientes permite selecionar clientes mais eficazes para o processo de treinamento. De fato, o *MetricSelection* permite escolher menos clientes para realizar o treinamento, enquanto atinge acurácia similar a escolha aleatória de uma proporção de clientes. Importante ressaltar que todos os clientes receberam amostras para treinamento uniformemente distribuídas (IID), justificando os bons resultados alcançados pela escolha aleatória de clientes. Em cenários não-IID, métricas para definição de seleção de clientes podem apresentar acurácia ainda maior do que a seleção aleatória por considerar a contribuição de cada cliente ao modelo. Ressalta-se que a avaliação de distribuição de dados nos clientes está fora do escopo deste trabalho e é deixado como trabalho futuro.

5. Conclusão e Trabalhos Futuros

Neste artigo foram apresentados os algoritmos *FedSketch* e *MetricSelection* para aumentar a privacidade, a eficiência na comunicação de dados e a convergência dos modelos no aprendizado federado. Foi observado como resultado principal que, mesmo com compressões de até 73 vezes, o algoritmo *FedSketch* atingiu resultados similares ao aprendizado federado tradicional. Por outro lado, o algoritmo *MetricSelection* conseguiu uma redução do número médio de clientes selecionados no processo de treinamento.

Como trabalhos futuros, pretende-se testar esses algoritmos em problemas distintos, tais como previsão em séries temporais e com distribuições não-IID. Além disso, pretende-se adaptar o algoritmo *MetricSelection* para utilização em cenários em que há clientes maliciosos no processo de aprendizado federado, tornando-se necessário identificá-los para garantir uma boa convergência dos modelos.

Agradecimentos

Este trabalho possui financiamento parcial de: CNPq, CAPES (Finance Code #001), Fapes (#2023/ RWXSZ, #2022/ ZQX6) e Fapesp/ MCTI/ CGI.br (#2020/ 05182-3).

Referências

Abdel-Basset, M., Hawash, H., and Moustafa, N. (2022). Toward privacy preserving federated learning in internet of vehicular things: Challenges and future directions. *IEEE Consumer Electronics Magazine*, 11(6):56–66.

Dwork, C. (2006). Differential privacy. In Bugliesi, M., Preneel, B., Sassone, V., and Wegener, I., editors, *Automata, Languages and Programming*, pages 1–12, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fu, L. et al. (2022). Client selection in federated learning: Principles, challenges, and opportunities. *ArXiv*, abs/2211.01549.

Haddadpour, F. et al. (2020). Fedsketch: Communication-efficient and private federated learning via sketching. *arXiv*, abs/2008.04975.

Husnoo, M. A. et al. (2022). FedREP: Towards horizontal federated load forecasting for retail energy providers. In *Asia-Pacific Power and Energy Engineering Conference (APPEEC)*. IEEE.

Isik-Polat, E., Polat, G., and Kocyigit, A. (2023). Arfed: Attack-resistant federated averaging based on outlier elimination. *Future Generation Computer Systems*, 141:626–650.

Joseph Near, D. D. (2023). Guidelines for evaluating differential privacy guarantees. Technical Report NIST SP 800-226 ipd, National Institute of Standards and Technology, Gaithersburg, MD.

Konečnỳ, J. et al. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv:1610.05492*.

Lecun, Y. et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, T., Liu, Z., Sekar, V., and Smith, V. (2019). Privacy for free: Communication-efficient learning with differential privacy using sketches. *ArXiv*, abs/1911.00972.

Liu, P., Xu, X., and Wang, W. (2022). Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives. *Cybersecurity*, 5.

McMahan, H. B. et al. (2016). Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*.

Nishio, T. and Yonetani, R. (2019). Client selection for federated learning with heterogeneous resources in mobile edge. In *International Conference on Communications (ICC)*, pages 1–7.

Smith, A., Song, S., and Guha Thakurta, A. (2020). The flajolet-martin sketch itself preserves differential privacy: Private counting with minimal space. *Advances in Neural Information Processing Systems*, 33:19561–19572.

Souza, A. et al. (2023). Dispositivos, eu escolho vocês: Seleção de clientes adaptativa para comunicação eficiente em aprendizado federado. In *XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Porto Alegre, RS, Brasil. SBC.

Wang, H. et al. (2022). Federated spatio-temporal traffic flow prediction based on graph convolutional network. In *2022 14th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 221–225.

Wei, K. et al. (2020). Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Trans on Information Forensics and Security*, 15:3454–3469.

Zhang, T. et al. (2022). Federated learning for internet of things: Applications, challenges, and opportunities.