O Que Não é Visto, Não é Lembrado: Aprendizado Federado Eficiente com Criptografia Homomórfica

Yuri Dimitre D. Faria[1], **Luiz F. Bittencourt**[1], **Leandro Villas**[1], **Allan M. de Souza**[1]

[1]Instituto de Computação - Universidade Estadual de Campinas

y218172@dac.unicamp.br, {bit,lvillas,allanms}@unicamp.br

Abstract. *Cross-silo federated learning (FL) allows multiple institutions to collaborate in training global models without directly sharing sensitive data. However, even without explicit data sharing, there are significant security risks, such as the possibility of gradient inversion attacks, which allow the reconstruction of private data from local client updates. To mitigate these risks, homomorphic encryption (HE) techniques have been widely explored, allowing operations on encrypted data without the need for decryption. However, the use of HE is accompanied by high computational and communication overhead, making its practical application difficult. In this context, this work proposes NVNL-FL, an efficient technique that combines packet packing and packet sparsification methods using a sliding window-based selection method, substantially reducing overhead, ensuring privacy, and maintaining model accuracy even in scenarios with heterogeneous (non-IID) data. Experimental results demonstrate that NVNL-FL overcomes limitations of existing methods, balancing efficiency, security, and performance. The full code for NVNL-FL is available on GitHub.*

Resumo. *O aprendizado federado (FL) cross-silo permite que múltiplas instituições colaborem no treinamento de modelos globais sem compartilhar dados sensíveis diretamente. No entanto, mesmo sem o compartilhamento explícito de dados, há riscos significativos de segurança, como ataques de inversão de gradientes, que permitem a reconstrução de dados privados a partir de atualizações locais dos clientes Para mitigar esses riscos, técnicas de criptografia homomórfica (HE) têm sido amplamente estudada, ao permitir operações em dados cifrados sem necessidade de decifração. Contudo, o uso de HE é acompanhado por um elevado overhead computacional e de comunicação, dificultando sua aplicação prática. Neste contexto, este trabalho propõe o NVNL-FL, uma abordagem eficiente que combina métodos de empacotamento e esparsificação de pacotes utilizando um método de seleção baseada em janelas deslizantes, reduzindo substancialmente o overhead, garantindo privacidade e mantendo a acurácia do modelo, mesmo em cenários com dados heterogêneos (não-IID). Os resultados experimentais demonstram que o NVNL-FL supera as limitações de métodos existentes, equilibrando eficiência, segurança e desempenho. O código completo do NVNL-FL está disponível no GitHub*[1]

1. Introdução

Cross-silo Federated Learning (FL) [Yang et al. 2019] é um paradigma de aprendizado distribuído promissor, o qual permite que múltiplas instituições (e.g., bancos ou hospitais)

[1]https://github.com/Numb4r/NVNL-FL

colaborem no treinamento de um modelo global como clientes. O paradigma *cross-silo* FL, tipicamente, tem presente um servidor central que orquestra os clientes e executa a agregação das atualizações locais (e.g., gradientes, parâmetros do modelo) em rodadas de sincronização [Capanema et al. 2025]. Esse paradigma é utilizado para que não seja necessário o compartilhamento de informações privadas das instituições participantes, que podem conter dados sensíveis, confidenciais ou regulados: cada participante envia apenas gradientes ou pesos do seu modelo de aprendizado treinado localmente. Ainda que esse sistema não revele os dados dos clientes de forma explicita, já foi demonstrado que agentes maliciosos, a partir do modelo global agregado, podem ser capazes de inferir informações que clientes utilizaram em seus treinamentos locais [Geiping et al. 2020].

Para mitigar o risco de vazamento de dados, soluções baseadas em criptografia homomórfica (*Homomorphic Encryption* – HE) têm sido exploradas para FL [Zhang et al. 2021]. A HE é uma técnica criptográfica que permite a execução de operações diretamente sobre dados criptografados, sem a necessidade de prévia decifragem. Em um cenário *cross-silo*, essa abordagem garante um alto nível de privacidade sem comprometer a acurácia do modelo [Li et al. 2023]. Com HE, os clientes podem cifrar suas atualizações locais de forma que o servidor, e outros clientes ou agentes externos, não consigam inferir as informações do criptogramas. Para o servidor de agregação as operações continuam inalteradas, já que a característica de homomorfismo permite operações aritméticas diretamente nos dados cifrados. Porém, técnicas de HE introduzem *overheads* computacionais e de comunicação significativos, já que realizam operações criptográficas intensas (e.g., multiplicação modular e redução polinomial) e geram criptogramas que têm magnitude maior que textos simples. Esquemas como Paillier [Paillier 1999] e CKKS [Cheon et al. 2017] aplicam operações intensas que podem resultar em uma latência de 24 a 81 minutos para cifrar uma CNN (*Convolutional Neural Network*) com 1,2 milhões de parâmetros, enquanto uma iteração de treinamento com texto simples demora 13 segundos, além do criptograma aumentar de 0,6GB para até 483GB, o que torna a latência de comunicação proibitiva [Jiang et al. 2021].

As principais abordagens para reduzir o *overhead* introduzido pela HE em FL incluem: *(i)* a redução da quantidade de gradientes ou pesos que precisam ser cifrados; e *(ii)* a agregação de múltiplos pesos em um único criptograma, técnica conhecida como empacotamento ou *batch*. Ambas as estratégias têm como objetivo diminuir o tamanho do criptograma final, reduzindo, assim, tanto o custo computacional para sua geração quanto a quantidade de dados a serem transmitidos [Choi et al. 2024].

Na primeira abordagem, métodos de *esparsificação* podem ser empregados. Esses métodos aplicam uma *máscara* de 0s e 1s com base na relevância dos neurônios do modelo [Yan et al. 2024]. Assim, apenas os pesos dos neurônios considerados relevantes para o modelo são cifrados. No entanto, diferentes clientes podem atribuir relevâncias distintas aos neurônios do modelo, dependendo de seus respectivos dados locais, o que pode resultar no desalinhamento dos pesos durante o processo de agregação [Tenison et al. 2023]. Como o servidor realiza a agregação com base apenas nos criptogramas compartilhados por cada cliente, sem conhecimento das posições dos neurônios no modelo global, as abordagens tradicionais falham em garantir uma agregação precisa devido ao desalinhamento causado pelas máscaras individuais dos clientes (i.e., *esparsificação*) [Yan et al. 2024]. Portanto, soluções mais robustas e sofisticadas precisam ser desenvolvidas para lidar com esse problema.

Por outro lado, a abordagem de empacotamento combina múltiplos valores (i.e., pesos ou gradientes do modelo) em um único "pacote", que é então cifrado. Como o número de pacotes gerados é menor do que o de gradientes a serem cifrados, o *overhead* imposto pela HE é reduzido. Além disso, é possível aplicar esparsificação nos pacotes, o que não apenas diminui ainda mais a quantidade de dados a serem cifrados, mas também mitiga o problema de desalinhamento causado pela esparsificação de neurônios durante a agregação [Jiang et al. 2021]. Uma abordagem de esparsificação de pacotes bastante utilizada é a Top-k, que seleciona os $k\%$ pacotes que atendem a um critério específico (e.g., pacotes cuja magnitude excede a média) [Yan et al. 2024]. No entanto, caso os pacotes não sejam selecionados adequadamente, a esparsificação pode comprometer ou até mesmo impedir a convergência do modelo, especialmente em cenários onde os dados são não balanceados e distribuídos de forma heterogênea [Xu et al. 2021, Souza et al. 2023, de Souza et al. 2024].

Nesse contexto, este trabalho introduz NVNL-FL (Não é Visto Não é Lembrado), uma solução eficiente de baixo *overhead* para FL com HE. Para isso, NVNL-FL combina métodos de empacotamento e esparsificação com uma janela deslizante para selecionar os pacotes a serem cifrados considerando, reduzindo o *overhead* enquanto permite a convergência do modelo mesmo em cenários não-IID. Os resultados mostram um ganho de $40\% - 43\%$ de acurácia em cenários não-IID em relação a técnica Top-k, e uma redução de tempo de execução de $17,63 - 31,19$ segundos em relação a técnica Batchcrypt.

Este trabalho está organizado da seguinte forma. A Seção 3 apresenta a revisão da literatura sobre HE para FL destacando vantagens e limitações das soluções apresentadas. A Seção 4 apresenta uma visão geral da NVNL-FL descrevendo o método de seleção dos pacotes a serem cifrados. A Seção 5 descreve a análise de desempenho da solução proposta em relação literatura considerando os impactos no desempenho do modelo e *overhead* produzido. Por fim, a Seção 6 apresenta a conclusão e trabalhos futuros.

2. Conceitos Básicos

Soluções de aprendizado federado baseadas em criptografia homomórfica permitem operações de agregação ou treinamento em criptogramas, sem a necessidade de revelar informações do modelo [Phong et al. 2018]. Seja x um texto simples e m_x o resultado de uma operação $m_x = m_1 + m_2$. Seja $Enc(x)$ a função para gerar o criptograma e $Dec(x)$ para retornar o criptograma para o texto simples x. Se a função $Enc(x)$ é uma função homomórfica, então $Enc(m_x) = Enc(m_1) + Enc(m_2)$. Assim, se aplicarmos a função $Enc(\cdot)$ a dois textos simples, e aplicar a função $Dec(\cdot)$ para obter o resultado da computação:

$$Enc(z) = Enc(x) + Enc(y) \rightarrow Dec(Enc(z)) = Dec(Enc(x) + Enc(y)) \rightarrow z = x + y \quad (1)$$

De modo geral, os esquemas de HE de Paillier [Paillier 1999], BFV [Fan and Vercauteren 2012] e CKKS [Cheon et al. 2017] são os mais utilizados em conjunto a FL. Paillier permite somente operações aditivas em criptogramas, enquanto BFV e CKKS permitem tanto adição quanto multiplicação. O esquema BFV e Paillier opera em criptogramas gerados a partir de números inteiros, enquanto CKKS opera com números reais e complexos.

Para reduzir o *overhead* introduzido pela HE no FL, diversas soluções baseiam-se no empacotamento dos parâmetros para reduzir a quantidade de criptogramas ge-

rados. Assim, variações de empacotamento de Paillier, BFV e CKKS são apresentadas [Yan et al. 2024]. Entretanto, somente com o empacotamento, tais soluções ainda podem ser ineficientes; assim, outras abordagens combinam o empacotamento com esparsificação de pacotes [Yan et al. 2024, Jiang et al. 2021], entretanto, essas podem limitar a convergência dos modelos.

3. Trabalhos Relacionados

O artigo de Zhang *et al.* [Zhang et al. 2020] apresenta o BatchCrypt, um método para otimizar a criptografia homomórfica no *cross-silo* FL usando o esquema de Paillier. A abordagem reduz o custo computacional e o volume de dados transferidos ao representar gradientes como inteiros de baixa precisão (8, 16 ou 32 bits) antes de codificá-los em números inteiros longos. Essa estratégia permite cifrar vários gradientes simultaneamente e preservar a aditividade da criptografia por meio de uma codificação baseada em complemento de dois com bits de sinal duplo. Embora diminua o *overhead* de comunicação, a técnica introduz latência adicional devido ao tempo necessário para quantização e cifragem.

Em seu artigo, Yan *et al.* [Yan et al. 2024] apresentam FedPHE, uma solução para *cross-silo* FL com HE baseado em empacotamento e esparsificação utilizando CKKS. No FedPHE, cada cliente vetoriza os parâmetros do modelo, organizado-os em pacotes de tamanho fixo e seleciona $k\%$ desses pacotes para serem cifrados e enviados para o servidor de agregação. Esse processo tem como base dois pontos principais, o algoritmo de escolha Top-k com um limiar calculado conforme a magnitude média dos pacotes (i.e., soma de todos os parâmetros presentes no pacote) e, a criação de uma máscara binaria, indicando se o pacote será transmitido (1) ou não (0), juntamente enviada ao servidor. Assim, o servidor utiliza as macaras recebidas para alinhar os pacotes encriptados provenientes de diferentes clientes, garantindo que somente os pacotes selecionados participem da agregação homomórfica. Entretanto, a abordagem de seleção de pacotes baseada em limiar não atinge desempenho satisfatório em cenários onde os dados são não-IID reduzindo ou até mesmo limitando a convergência do modelo.

O método FLASHE, desenvolvido por Jiang et al. [Jiang et al. 2021], propõe uma abordagem para FL *cross-silo* utilizando HE, reduzindo os custos associados aos esquemas HE tradicionais ao evitar criptografia assimétrica e priorizar operações de adição homomórfica, suficientes para a maioria dos algoritmos de agregação de modelos.O método utiliza máscaras aleatórias para garantir segurança eficientemente, empregando um sistema de mascaramento duplo que reduz custos computacionais e o tamanho dos dados criptografados, mantendo a segurança semântica. Além disso, FLASHE é otimizado para cenários com esparsificação, onde somente gradientes locais mais significativos são transmitidos. O esquema compacta as atualizações locais antes da criptografia, reduzindo o tráfego de comunicação e garantindo compatibilidade com esparsificação, algo que soluções de criptografia por lote não conseguem alcançar.

Os trabalhos analisados apresentam avanços significativos em HE para FL, mas têm limitações importantes. O FLASHE enfrenta desafios de escalabilidade devido ao custo computacional elevado na geração e remoção de máscaras aleatórias. O BatchCrypt sofre com perdas de precisão causadas pela quantização dos gradientes e depende de pré-processamentos adicionais, aumentando a complexidade e acrescenta *overhead* em tempo de processamento. Já o FedPHE, apresenta convergência limitada em cenários de dados

não-IID devido ao método de seleção dos pacotes baseados em limiar. Essas limitações indicam a necessidade de melhorias para ampliar a eficiência, a escalabilidade e a robustez desses métodos em cenários reais de FL.

4. NVNL-FL: Não é Visto Não é Lembrado

O principal objetivo do NVNL-FL é garantir a privacidade em cenários de *cross-silo* FL utilizando HE de forma eficiente, reduzindo o *overhead* de comunicação e computação enquanto mantém o desempenho do modelo. Para isso, NVNL-FL combina métodos de empacotamento e de esparsificação de pacotes para gerar os criptograma, consequentemente reduzindo a quantidade de dados que precisam ser cifrados e compartilhados com o servidor. A descrição detalhada do processo completo implementado por NVNL-FL bem como o método de esparsificação e pacotes são descritos nas seções a seguir.

4.1. Definição do Problema

Seja $\mathcal{N} = 1, 2, \ldots, N$ o conjunto de N clientes em um cenário de FL *cross-silo*, onde os N clientes treinam um modelo de forma colaborativa sem compartilhamento de dados. Cada cliente i tem uma base de dados d_i de forma que $\mathcal{D} = \bigcup_{i \in \mathcal{N}} d_i$ representa a distribuição global dos dados. A cada rodada de comunicação t, o servidor seleciona um subconjunto de clientes $\mathcal{S} \subseteq \mathcal{N}$ e envia um modelo w_g^t representado pelos pesos do modelo w global agregado na rodada de comunicação $t-1$. Assim, os clientes selecionados recebem o modelo global w_g^t e realizam o treinamento em seus dados d_i. Então, cada cliente i retorna as atualizações do modelo para o servidor. No final da rodada de comunicação t, o modelo global é atualizado através da média ponderada dos pesos dos neurônios correspondentes nos modelos enviados pelos clientes de acordo com a Equação 2.

$$w_g^{t+1} = \sum_{i=1}^{|\mathcal{S}|} \frac{|d_i|}{|\mathcal{D}|} w_i^t \tag{2}$$

Portanto, o objetivo principal em FL é minimizar a seguinte função objetivo $F(w_g)$:

$$\min_w F(w_g) \quad \text{onde} \quad F(w_g) = \sum_{i=1}^{|\mathcal{S}|} \frac{|d_i|}{|\mathcal{D}|} \mathcal{L}(w_g; x_i, y_i) \tag{3}$$

onde w_g é o modelo global, $\mathcal{L}$ é a função de perda, x_i e y_i, respectivamente, são entradas de dados e rótulos do conjunto de dados d_i para cada cliente.

Embora o FL evite o compartilhamento direto de dados para treinar modelos colaborativos, existe o risco de que os dados possam ser recuperados indiretamente por meio de técnicas de inversão de gradientes exploradas por agentes mal-intencionados [Yan et al. 2024]. Portanto, a HE torna-se essencial para "**esconder**" os modelos compartilhados de entidades mal-intencionadas, evitando consequentemente a **recuperação** dos dados dos clientes através de ataques de inversão de gradientes.

4.2. Modelo de Ameaça

Em um modelo de ameaça *honesto-mas-curioso* (também conhecido como *semi-honesto*) em FL, o servidor adere ao protocolo definido corretamente, mas permanece curioso sobre os dados compartilhados pelos participantes. O servidor observa os gradientes ou

atualizações de modelo enviados por clientes para reconstruir informações confidenciais usando técnicas como ataques de inversão de gradiente. Este modelo de ameaça é particularmente relevante em cenários onde o servidor é gerenciado por uma parte potencialmente não confiável, mas os clientes dependem do servidor para gerenciar o processo de FL.

Para proteger contra essas ameaças, HE é empregado para cifrar gradientes ou atualizações antes que sejam enviados ao servidor, garantindo que os cálculos de agregação dos gradientes possam ser realizados diretamente em dados criptografados, impedindo que o servidor acesse gradientes brutos e, assim, eliminando o risco de ataques de inversão de gradiente. Simultaneamente, os protocolos *Transport Layer Security* (TLS) ou *Secure Sockets Layer* (SSL) protegem os canais de comunicação entre clientes e o servidor, impedindo que adversários externos interceptem ou adulterem os dados durante a transmissão. Os protocolos TLS/SSL criptografam dados em trânsito, garantindo que apenas partes autenticadas no processo de aprendizado federado possam acessar as atualizações transmitidas. Juntos, HE e TLS/SSL formam uma defesa robusta, sendo que a HE protege os dados da curiosidade do servidor enquanto TLS/SSL impede a espionagem e a interceptação por agentes externos, garantindo um ambiente de aprendizado federado seguro e que preserva a privacidade.

4.3. Visão Geral do NVNL-FL

Diferentemente de FedPHE e BatchCrypt, que utilizam métodos estáticos (e.g., Top-K) ou quantização agressiva, o NVNL-FL introduz um método dinâmico de seleção de pacotes baseado em janela deslizante, garantindo maior adaptabilidade em cenários não-IID.

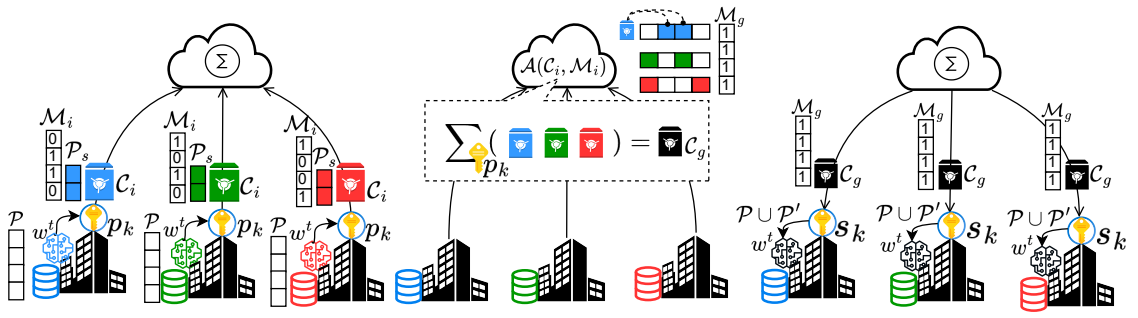


Figura 1. Visão geral do NVNL-FLdescrevendo etapas de funcionamento, incluindo: empacotamento, esparsificação de pacotes, cifragem, alinhamento, agregação e decifragem.

De acordo com o problema e o modelo de ameaça definido, o funcionamento completo do NVNL-FL é ilustrado na Figura 1. Nesse cenário, os clientes possuem um par de chaves secreta e pública (s_k, p_k) compartilhadas utilizando algum algoritmo de compartilhamento de chaves conhecido. Cada cliente $i \in \mathcal{N}$ realiza o treinamento do modelo de forma convencional, faz o empacotamento dos pesos do modelo $\mathcal{P} = Packing(w, size)$ baseado em um tamanho $size$ predefinido, calcula uma máscara $\mathcal{M}_i$ de acordo com um algoritmo de seleção (que define quais pacotes serão cifrados). Portanto, utilizando a chave pública p_k cada cliente cifra os pacotes selecionados definidos como $\mathcal{P}_s = \mathcal{P} \oplus \mathcal{M}_i$, em seguida, compartilha tanto o modelo cifrado $\mathcal{C}_i = Enc_{p_k}(p)\ \forall\ p \in \mathcal{P}_s$, quanto o tamanho de seu *dataset* $|D_i|$ (necessário para a agregação ponderada) e a máscara $\mathcal{M}$ para realizar o alinhamento dos pacotes durante a agregação.

O servidor, por sua vez, recebe os criptogramas $\mathcal{C}_i$ enviados pelos por cada cliente $i \in \mathcal{N}$, o tamanho $|D_i|$ do *dataset* do cliente e sua máscara $\mathcal{M}_i$. Antes de realizar a agregação ponderada, o servidor realiza o alinhamento dos criptogramas de acordo com a máscara recebida de cada cliente $\mathcal{A}(\mathcal{C}_i, \mathcal{M}_i)$ e, utilizando a chave pública p_k, realiza a agregação ponderada dos criptogramas alinhados e também das máscaras, resultando em um novo modelo cifrado $\mathcal{C}_g$ e uma máscara agregada $\mathcal{M}_g$. Esse novo modelo e a máscara são então enviados de volta aos clientes. Por fim, os clientes recebem o modelo cifrado atualizado e a máscara global, decifram os pacotes agregados $\mathcal{P}' = Dec_{s_k}(\mathcal{C}_g)$ e de acordo com a máscara global realizam a alteração dos pacotes, $\mathcal{P}_{al} \leftarrow \mathcal{P}' \oplus \mathcal{M}_g$, gerando o novo modelo no cliente baseados nos pacotes dos clientes $\mathcal{P}$ em conjunto com os pacotes agregados recebidos $\mathcal{P}'$, tal que $w^t = \mathcal{P} \cup \mathcal{P}_{al}$ incorporando o conhecimento compartilhado. Com isso, inicia-se uma nova rodada de treinamento.

O Algoritmo 1 descreve as operações realizadas pelo servidor para alinhamento e agregação de criptogramas com HE. O algoritmo recebe como entradas: o conjunto de clientes $\mathcal{N}$ participantes de federação; a quantidade de rodadas T; a porcentagem $frac$ de clientes que serão selecionados a cada rodada; e a chave pública p_k para realizar as operações homomórficas nos criptogramas. Inicialmente, o servidor seleciona um cliente aleatório para inicializar o sistema, onde o cliente retorna todos os pacotes representando todos os pesos do modelo. Assim, na primeira rodada $\mathcal{C}_g$ é representado por todos os pacotes do modelo (cifrados), assim $\mathcal{M}_g$ é [1, 1, ..., 1]. Em seguida, para cada rodada $t \in [1, 2, \ldots, T]$ o servidor seleciona um subconjunto de clientes aleatoriamente de acordo com a porcentagem $frac$ (Linha 6). Então, o servidor envia para os clientes os criptogramas $\mathcal{C}_g$ e a máscara $\mathcal{M}_g$ para que os clientes possam realizar o treinamento e compartilhar um novo modelo (Linha 8).

Algorithm 1: Algoritmo executado pelo servidor para alinhamento e agregação de criptogramas utilizando HE

```
   input : 𝒩 /* Conjunto de clientes na federação                    */
         1 T /* Número máximo de rodadas de comunicação              */
         2 frac /* Porcentagem de clientes selecionados              */
         3 p_k /* chave pública HE                                   */
 4 Servidor (𝒩, T, frac):
       /* Seleciona cliente aleatório para inicializar modelo       */
       /* Na primeira rodada 𝒞_g é representado por todos os
          pacotes do modelo, assim ℳ_g é [1, 1, ..., 1]            */
 5     for t ∈ {1, 2, 3, ... T} do
           // Seleciona clientes de acordo com a porcentagem frac
 6         𝒮 ← RandomSamplig(𝒩, frac);
 7         foreach i ∈ 𝒮 do
               // Envia modelo para os clientes i ∈ 𝒮
 8             𝒞_i, ℳ_i, |d_i| ← ClientLocalTrain(i, w^g, t);
               // Alinha criptogramas 𝒞_i de acordo com a máscara ℳ_i
 9             𝒞_i ← 𝒜(𝒞_i, ℳ_i)
10         𝒞_g, ℳ_g ←Aggregate(𝒞_i, ℳ_i, D_i, p_k)
```

Ao final do treinamento, cada cliente i retorna para o servidor os criptogramas $\mathcal{C}_i$ (i.e., cifragem dos pacotes selecionados), a máscara para o alinhamento dos pacotes $\mathcal{M}_i$ e o tamanho de seu *dataset* $|d_i|$ para agregação ponderada. Assim, o servidor alinha os criptogramas $\mathcal{C}_i$ de acordo com a máscara $\mathcal{M}_i$ para realizar a agregação de forma

adequada na Linha 9 (i.e., para que todos os criptogramas sejam posicionados corretamente em suas partes originais do modelo). A agregação dos criptogramas ocorre a Linha 10 o servidor agrega os criptogramas e também as máscaras dos clientes utilizando a chave pública p_k. Em particular, a agregação ponderada é realizada da seguinte forma $\mathcal{C}_g = \mathcal{C}_g \cdot \left(\frac{|D_i|}{|\mathcal{D}|} \cdot \mathcal{C}_i\right) \ \forall i \in \mathcal{S}$. Dessa forma, um novo criptograma $\mathcal{C}_g$ representando os pacotes agregados dos clientes e a máscara informando quais pacotes foram agregados $\mathcal{M}_g$ são construídos e serão enviados novamente para os clientes para continuidade no treinamento. É importante destacar, que em nenhum momento o servidor tem acesso aos dados dos modelos dos clientes, consequentemente inviabilizando ataques de inversão para recuperar dados de algum cliente. O Algoritmo 2 descreve as operações realizadas

Algorithm 2: Algoritmo executado pelos clientes para treinamento, empacotamento, seleção de pacotes, cifragem e decifragem utilizando HE

```
   input : /* D_i dataset do cliente i                                  */
   /* s_k Chave secreta HE                                             */
   /* p_k chave pública HE                                             */
   /* size Tamanho dos pacotes                                         */
   /* k Quantidade de pacotes a serem selecionados                     */
   /* stride Deslocamento utilizado na seleção de pacotes              */
1  ClientLocalTrain (C_g, M_g, t):
       // Decifra criptogramas C_g com chave secreta s_k
2      P' ← DEC(C_g, s_k);
       // Alinha pacotes baseado na máscara agregada M_g
3      P' ← A(P', M_g);
       // Combina pacotes agregados com pacotes locais
4      P ←MERGE(P, P');
       // Converte pacotes para estrutura do modelo
5      w_i ←PACKSTOWEIGHTS(P);
       // Treina modelo com dados locais e atualiza pesos w
6      for e ∈ #Epocs do
7          for (x, y) ∈ D_i do
8              w_i ← w_i · η∇L(w; (x, y));
       // Empacota pesos, seleciona pacotes P_s e máscara M_i
9      P ←PACKING(w, size);
10     P_s, M_i ← SLIDEWINDOWSELECTION(P, k, t);
       // Encripta P_s com CKKS e chave a pública p_k
11     C_i ←ENC(P_s, p_k);
12     return C_i, M_i, |D_i|;
13 SlideWindowSelection (P, k, t, stride):
14     M_i ← [0, 0, ..., 0];
15     for p ∈ {1, 2, ..., |P|} do
16         if p ∈ [t · stride, ..., t · stride + k] then
17             M_i[p] ← 1;
18     P_s ← P ⊕ M_i;
19     return P_s, M_i;
```

pelo cliente ao receber os criptogramas e mascaras agregados do servidor. O algoritmo recebe como entrada: D_i o *dataset* do cliente; a chave secreta s_k e pública p_k para realizar as operações de HE; a variável $size$ que define o tamanho dos pacotes (i.e., quantidade de pesos em cada pacote); a quantidade de pacotes k que serão selecionados; e $stride$

uma variável informando o deslocamento utilizado pelo método de seleção de pacotes. Cada cliente recebe os criptogramas e máscara agregados $\mathcal{C}_g$ e $\mathcal{M}_g$, respectivamente. Então, o cliente decifra os pacotes utilizando a função de decifragem e a chave secreta $\mathcal{P}' \leftarrow Dec(\mathcal{C}_g, s_k)$ Linha 2. Em seguida, cada cliente precisa também alinhar os pacotes agregados baseado na máscara $\mathcal{M}_g$, atualizando $\mathcal{P}'$ e também combinar com seus pacotes anteriores $\mathcal{P}$ (Linhas 3-4). Os pacotes então são convertidos novamente para a estrutura de pesos do modelo (Linha 5).

O treinamento do modelo é realizado normalmente (Linha 8) para cada *batch* dos dados do cliente por uma quantidade definida de épocas. Ao final do treinamento, o cliente então realiza o empacotamento dos pesos do modelo $\mathcal{P}$ de acordo com a quantidade de pesos por pacote $size$. É importante destacar que nesse processo também ocorre *padding* quando a quantidade de pesos disponíveis é menor que o tamanho definido do pacote. Portanto, o cliente seleciona os pacotes que serão cifrados e compartilhados com o servidor utilizando um método de *janela deslizante* baseado na rodada de comunicação t, no tamanho da janela k e também no tamanho do deslocamento realizado por rodada $stride$ (Linha 10). O método de seleção baseado em janela deslizante é descrito entre as Linhas 12-19.

O método de janela deslizante foi definido para sincronizar o treinamento colaborativo dos neurônios do modelo e também para garantir que todos os neurônios serão compartilhados entre os clientes: quanto maior a sobreposição e compartilhamento de neurônios, maior é o aprendizado compartilhado entre os clientes. Dessa forma, superando limitações de outras abordagens, como por exemplo a Top-k que apresenta desempenho limitado em dados não-IID, onde pode não só haver pouca sobreposição na agregação dos neurônios, como também não compartilhar todos os neurônios do modelo, uma vez que os pacotes Top-k pode ser fixa [Yan et al. 2024].

Por fim, os clientes cifram os pacotes selecionados $\mathcal{P}_s$ com o método CKKS e sua chave pública, enviando ao servidor os pacotes cifrados $\mathcal{C}_i$, a máscara $\mathcal{M}_i$ (que indica os pacotes selecionados) e o tamanho do dataset D_i para agregação ponderada. Vale destacar que todo o processo de comunicação e agregação é feito com os dados cifrados de cada cliente. Isso garante um maior grau de segurança a ataques de inversão, tornando o processo de inferir dados privados recuperados dos clientes tão difícil quanto quebrar a criptografia empregada.

5. Experimentos e Resultados

5.1. Ambiente de Avaliação

Um ambiente baseado em contêineres *docker* foi criado para a execução, onde a ideia principal é dividir as aplicações servidor e cliente em diferentes contêineres, que podem ser executados no mesmo *host* ou em diferentes máquinas. Para isso, usamos um *Docker Swarm* para a criação do *cluster*. Foram definidas configurações específicas para o contêiner do servidor e dos clientes utilizando *docker-compose*. Essa abordagem facilita a replicação, a configuração de testes e a execução em diferentes ambientes, necessitando apenas da instalação do *Docker*. Uma vez criado o arquivo *docker-compose*, o experimento pode ser iniciado com o comando: `docker-compose -f nome_do_arquivo_criado.yaml up`.

Os experimentos foram conduzidos em uma máquina com processador Ryzen 9 5900x e uma GPU RTX 3080ti. Para o treinamento e desenvolvimento dos modelos, foi

utilizado Tensorflow, enquanto para o desenvolvimento do protocolo de FL foi utilizado o *framework* Flower. Além disso, para a implementação da criptografia homomórfica, foi utilizada a biblioteca TenSEAL.

5.2. Datasets e Modelos

Foram utilizados dois *datasets* amplamente conhecidos: *(i)* MNIST; e *(ii)* FashionMNIST. Para o treinamento do MNIST foi utilizada uma *Deep Neural Network* (DNN) composta por camadas de 128, 64 e 32 neurônios com funções de ativação ReLU e uma camada de saída *softmax*. Por outro lado, para o treinamento do FashionMNIST foi utilizada a rede LeNet-5 [Xiao et al. 2017] composta por sete camadas, incluindo camadas convolucionais, de *pooling* (subamostragem) e totalmente conectadas, organizadas de forma hierárquica. Por fim, para o treinamento foi utilizado *Stochastic Gradient Descent* (SGD) como método para atualizar os pesos do modelo e a *Sparse Categorical Crossentropy* como função de perda.

5.3. Métricas

Para analisar o desempenho do modelo e também o *overhead* de processamento e comunicação das soluções, as seguintes métricas foram utilizadas: *(i)* Desempenho do modelo: acurácia média do modelo no conjunto de teste dos clientes calculados a cada rodada de comunicação. Com essa métrica é possível analisar o comportamento da aprendizagem do modelo ao longo do tempo; *(ii) Overhead* de Comunicação: quantidade total de dados transmitidos pelas soluções para realizar o treinamento do modelo. Essa métrica mostra o custo adicional de comunicação gerado pelas soluções de HE; e *(iii) Overhead* de treinamento: tempo médio gasto por rodada de comunicação das soluções. O tempo inclui as seguintes etapas: *(i)* tempo de treinamento; *(ii)* tempo de cifragem; *(iii)* tempo de decifragem.

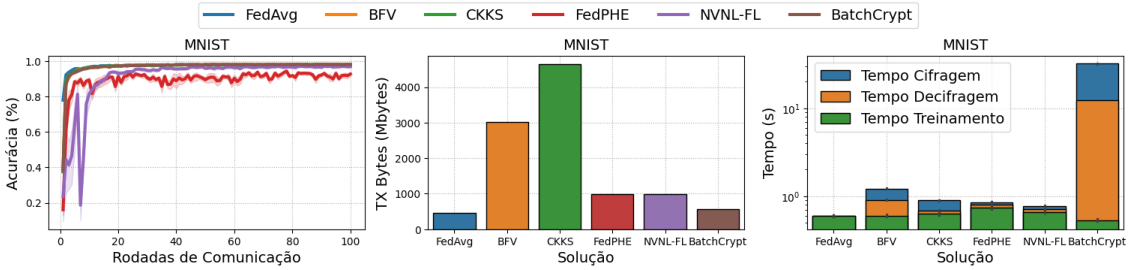
5.4. Configuração dos Experimentos

O número total de clientes foi definido como $\mathcal{N} = 10$, com $frac = 0,5$, ou seja, apenas 50% dos clientes são selecionados para o treinamento durante cada rodada de comunicação. Foram realizadas 100 rodadas de comunicação. A heterogeneidade de dados foi controlada pela distribuição de Dirichlet [Hsu et al. 2019] para permitir um ambiente IID com $\alpha = 1$ e também não-IID com $\alpha = 0.1$. Com isso, podemos avaliar o desempenho das soluções para ambos os cenários.

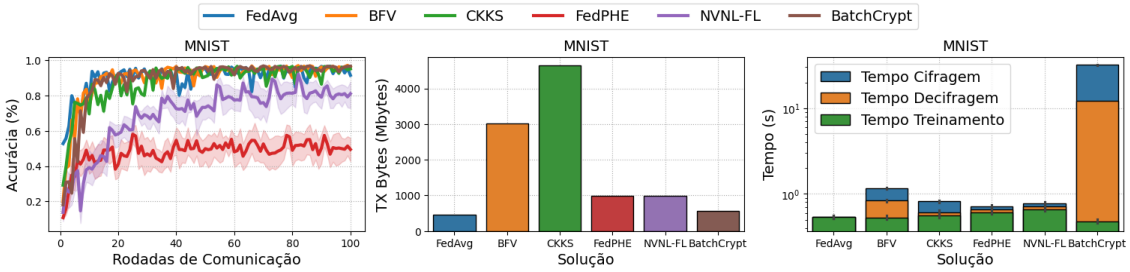
Para os métodos de criptografia homomórfica, no CKKS utilizamos grau do módulo do polinômio 8192 e tamanho dos bits dos módulos dos coeficientes $[60, 40, 40, 60]$, enquanto para o BFV utilizamos um polinômio de 8192 e módulo base de 1032193. Para o Paillier, foi configurado o tamanho do criptograma de 2048 *bits*. Além disso, para soluções de empacotamento foi utilizado $size = 2048$, ou seja, cada pacote contém informações de 2048 neurônios do modelo.

Para analisar o desempenho do NVNL-FL, comparamos com as seguintes técnicas da literatura: *(i)* FedAvg [McMahan et al. 2023], a qual implementa o processo tradicional de FL sem cifragem; *(ii)* CKKS, onde foi implementado o empacotamento com cifragem CKKS; *(iii)* BFV, representando o empacotamento com cifragem BFV; *(iv)* BatchCrypt [Zhang et al. 2020] que realiza o empacotamento e cifragem de Paillier; e *(v)* FedPHE [Yan et al. 2024] que realiza o empacotamento e seleciona $k\%$ dos pacotes utilizando um método de Top-k, baseado na magnitude dos pesos dos pacotes.

5.5. Análise dos Resultados



(a) IID $\alpha = 1$



(b) Não-IID $\alpha = 0.1$

Figura 2. Resultados de desempenho e *overhead* de comunicação e processamento das soluções no *dataset* MNIST em cenários IID e não-IID

A Figura 2 apresenta uma comparação do NVNL-FL em relação as soluções da literatura, considerando métricas de desempenho, *overhead* de comunicação e processamento para o *dataset* MNIST com dados IDD (Figura 2(a)) e não-IID (Figura 2(b)). É importante destacar que os gráficos para o FashionMNIST não foram apresentados por limitação de espaço, mas os mesmos estão resumidos na Tabela 1. Como pode ser observado, as soluções alcançam um desempenho similar em termos de acurácia para cenários IID (Figura 2(a)). Entretanto, ao analisar o *overhead* produzido pelas soluções, podemos observar o aumento da transmissão de dados e também do tempo de computação para as soluções baseadas em HE.

O FedAvg mostra o comportamento tradicional do sistema, fornecendo uma base para desempenho, comunicação e processamento. As soluções BFV e CKKS, apresentam um *overhead* de comunicação de 5 e 9 maior que o FedAvg, respectivamente. Apesar dessa soluções realizarem o empacotamento dos pesos do modelo, elas cifram todos os pacotes, assim não apresentam perda de desempenho, mas consequentemente aumentam muito a quantidade de dados a serem transmitidos entre clientes e servidor, o que pode limitar a escalabilidade dessas soluções. Considerando o tempo de processamento, podemos ver um aumento maior no BFV devido a necessidade da quantização dos pesos do modelo para realizar as operações do modelo. Assim, o BFV aumenta o tempo de processamento em mais de 100%, enquanto o CKKS aumenta até 50% dependendo do tamanho do modelo (veja resultados Tabela 1).

Por outro lado, o BatchCrypt também realiza o empacotamento e cifragem de todos os pacotes, porém produzindo um *overhead* de comunicação muito inferior as soluções BFV e CKKS, devido ao esquema de Paillier adotado para realizar a cifragem dos pacotes. Em particular o BatchCrypt aumenta aproximadamente apenas 25% da quantidade de dados a serem transmitidos, atingindo um desempenho similar ao FedAvg, tanto

Tabela 1. Análise dos Resultados do NVNL-FL em relação a literatura para os *datasets* MNIST e FashionMNIST com dados IDD e não-IID.

		MNIST					FashionMNIST				
Dados	**Soluções**	**Acurácia**	**TX Bytes**	**Tempo**	**OC**	**OP**	**Acurácia**	**TX Bytes**	**Tempo**	**OC**	**OP**
IID	**FedAvg**	0,97	449,49	0,59	0,00	0,00	0,90	272,25	2,06	0,00	0,00
	BFV	0,97	3009,08	1,21	5,69	1,05	0,90	1719,48	2,46	5,32	0,19
	CKKS	0,97	4651,85	0,89	9,35	0,51	0,90	2658,22	2,22	8,76	0,08
	BatchCrypt	0,97	571,77	32,14	0,27	53,47	0,88	317,68	20,35	0,17	8,88
	FedPHE	0,94	996,94	0,72	1,22	0,22	0,81	664,63	2,26	1,44	0,10
	NVNL-FL	0,96	996,95	0,76	1,22	0,29	0,85	664,63	2,36	1,44	0,15
Não-IID	**FedAvg**	0,97	449,49	0,53	0,00	0,00	0,84	272,25	1,69	0,00	0,00
	BFV	0,95	3009,08	1,15	5,69	1,17	0,79	1719,48	2,21	5,32	0,31
	CKKS	0,95	4651,85	0,82	9,35	0,55	0,81	2658,22	1,94	8,76	0,15
	BatchCrypt	0,95	571,77	31,96	0,27	59,30	0,83	317,68	19,99	0,17	10,83
	FedPHE	0,49	996,94	0,71	1,22	0,34	0,35	664,63	2,26	1,44	0,34
	NVNL-FL	0,92	996,95	0,77	1,22	0,45	0,75	664,63	2,36	1,44	0,40

*As unidades são acurácia (%), TX bytes (Mbytes), Tempo (s). Além disso, as métricas OC e OP representam o *overhead* de comunicação e processamento, respectivamente e são calculadas de acordo com o custo de comunicação (OC) e processamento (OP) do FedAvg.

para dados IID (Figura 2(a)) quanto não-IID (Figura 2(a)). Porém, quando analisamos o *overhead* de processamento produzido pelo BatchCrypt, vimos que pode aumentar em mais de 50 vezes o tempo de processamento por rodada de comunicação em relação ao FedAvg. Tal *overhead* é consequência dos métodos de quantização e cifragem utilizados pela solução, que apesar de produzir menor quantidade de dados para transmissão, são muito complexos e custos para o processamento. Dessa forma, o BatchCrypt pode introduzir um *overhead* indesejado para o sistema além de limitar sua aplicação em dispositivos com baixo poder computacional.

Ao contrário das soluções apresentadas o FedPHE realiza a seleção dos pacotes a serem cifrados com o método CKKS, assim reduzindo tanto a quantidade de dados a serem transmitidos quanto o tempo de processamento. Entretanto, o método utilizado para seleção dos pacotes é o Top-k baseado em magnitude dos pacotes. Essa abordagem pode limitar o desempenho do modelo em cenários de dados não-IID, pois os pacotes selecionados pelos os clientes pode se tornar fixos e disjuntas entre os clientes, limitando o compartilhamento de conhecimento entre os clientes (i.e., sem sobreposição durante a agregação dos pesos dos modelos no servidor). Dessa forma, o modelo não é capaz de atingir níveis satisfatórios de generalização e, consequentemente degradando seu desempenho. Em particular, quando analisamos o desempenho do FedPHE em cenários não-IID, podemos ver que seu desempenho não atinge nem 50% de acurácia para os *datasets* analisados (veja resultados Tabela 1)

A combinação de empacotamento com seleção de pacotes via janela deslizante permite ao NVNL-FL reduzir significativamente o overhead de comunicação e processamento, mantendo acurácia superior a 90% mesmo em cenários não-IID. Embora a convergência ocorra de forma mais lenta, o desempenho final é comparável às soluções da literatura. A sobrecarga de comunicação é apenas 1,2 vezes maior que a do FedAvg, e o overhead de processamento representa 40% das soluções baseadas em HE, tornando o NVNL-FL uma alternativa eficiente e promissora para aplicações reais que exigem privacidade e baixo custo computacional.

Os resultados completos podem ser visto na Tabela 1 para os dois *datasets* analisados bem como as configurações IID e não-IID. É importante destacar que os valores de *overhead* de comunicação (OC) e *overhead* de processamento (OP) utilizaram como base

a performance do FedAvg, assim esses valores apresentam o aumento na comunicação e tempo introduzido pelas soluções.

Com os resultados apresentados, conclui-se que: *(i)* NVNL-FL introduz um baixo *overhead* para o treinamento federado de modelos utilizando HE; *(ii)* aumenta a privacidade dos dado dos clientes evitando ataques de inversão para recuperação de informações devido as operações em cima de dados cifrados; *(iii)* o método de janela deslizante implementado permite o compartilhamento de conhecimento entre os clientes, garantindo o aprendizado do modelo mesmo em cenários não-IID.

6. Conclusão

Este trabalho introduz o NVNL-FL, uma solução inovadora e eficiente para FL *cross-silo* com criptografía homomórfica, capaz de maximizar a privacidade dos dados sem comprometer o desempenho. Ao combinar técnicas de empacotamento e esparsificação de pacotes com um método avançado de seleção baseado em janelas deslizantes, o NVNL-FL reduz significativamente o *overhead* computacional e de comunicação, atendendo às necessidades de segurança e eficiência em cenários desafiadores, como aqueles com dados heterogêneos (não-IID). Os resultados experimentais destacam a robustez e eficácia do NVNL-FL, comprovando sua capacidade de superar limitações de métodos existentes. A solução não apenas equilibra segurança, desempenho e eficiência, mas também estabelece um novo patamar para aplicações de aprendizado colaborativo em escala, oferecendo altos níveis de acurácia mesmo em condições adversas. Como direções futuras, planeja-se otimizar ainda mais as técnicas de seleção de pacotes, aprofundar a análise do overhead de comunicação em ambientes reais e explorar a escalabilidade da solução em testbeds.

7. Agradecimentos

Este projeto foi apoiado pelo Ministério da Ciência, Tecnologia e Inovações, com recursos da Lei nº 8.248, de 23 de outubro de 1991, no âmbito do PPI-SOFTEX, coordenado pela Softex e publicado Arquitetura Cognitiva (Fase 3), DOU 01245.003479/2024-10. Este trabalho foi parcialmente patrocinado pelo projeto #23/00673-7, São Paulo Instituto de Pesquisa (FAPESP), CNPq projeto #405940/2022-0 e CAPES projeto #88887.954253/2024-00 e PIND UNICAMP projeto #519.287

Disponibilidade de Artefato

Os artefatos do projeto, incluindo código fonte e *framework* para execução e configuração dos experimentos está disponível no GitHub[2]

Referências

Capanema, C. G. S., de Souza, A. M., da Costa, J. B. D., Silva, F. A., Villas, L. A., and Loureiro, A. A. F. (2025). A novel prediction technique for federated learning. *IEEE Transactions on Emerging Topics in Computing*, 13(1):5–21.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In Takagi, T. and Peyrin, T., editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham. Springer International Publishing.

[2]`https://github.com/Numb4r/NVNL-FL`

Choi, S., Patel, D., Zad Tootaghaj, D., Cao, L., Ahmed, F., and Sharma, P. (2024). Fednic: enhancing privacy-preserving federated learning via homomorphic encryption offload on smartnic. *Frontiers in Computer Science*, 6.

de Souza, A. M., Maciel, F., da Costa, J. B., Bittencourt, L. F., Cerqueira, E., Loureiro, A. A., and Villas, L. A. (2024). Adaptive client selection with personalization for communication efficient federated learning. *Ad Hoc Networks*, 157:103462.

Fan, J. and Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144.

Geiping, J., Bauermeister, H., Dröge, H., and Moeller, M. (2020). Inverting gradients – how easy is it to break privacy in federated learning?

Hsu, T.-M. H., Qi, H., and Brown, M. (2019). Measuring the effects of non-identical data distribution for federated visual classification.

Jiang, Z., Wang, W., and Liu, Y. (2021). Flashe: Additively symmetric homomorphic encryption for cross-silo federated learning.

Li, A., Peng, H., Zhang, L., Huang, J., Guo, Q., Yu, H., and Liu, Y. (2023). Fedsdg-fs: Efficient and secure feature selection for vertical federated learning.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2023). Communication-efficient learning of deep networks from decentralized data.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In Stern, J., editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg. Springer Berlin Heidelberg.

Phong, L. T., Aono, Y., Hayashi, T., Wang, L., and Moriai, S. (2018). Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345.

Souza, A., Bittencourt, L., Cerqueira, E., Loureiro, A., and Villas, L. (2023). Dispositivos, eu escolho vocês: Seleção de clientes adaptativa para comunicação eficiente em aprendizado federado. In *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 1–14, Porto Alegre, RS, Brasil. SBC.

Tenison, I., Sreeramadas, S. A., Mugunthan, V., Oyallon, E., Rish, I., and Belilovsky, E. (2023). Gradient masked averaging for federated learning.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Xu, W., Fan, H., Li, K., and Yang, K. (2021). Efficient batch homomorphic encryption for vertically federated xgboost.

Yan, N., Li, Y., Chen, J., Wang, X., Hong, J., He, K., and Wang, W. (2024). Efficient and straggler-resistant homomorphic encryption for heterogeneous federated learning. *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 791–800.

Yang, Q., Liu, Y., Chen, T., and Tong, Y. (2019). Federated machine learning: Concept and applications.

Zhang, C., Li, S., Xia, J., Wang, W., Yan, F., and Liu, Y. (2020). BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 493–506. USENIX Association.

Zhang, S., Li, Z., Chen, Q., Zheng, W., Leng, J., and Guo, M. (2021). Dubhe: Towards data unbiasedness with homomorphic encryption in federated learning client selection.