Capítulo 1

Ataques em Aprendizado Federado: Impactos Práticos e Estratégias de Mitigação

Helio N. Cunha Neto (UERJ), Carlos Henrique Nunes (UERJ),
Ricardo Lundgren (UFF), Raphael Jorge B. Ortolan (UERJ),
Luiz H. S. Ladeira (UERJ), Ian Vilar Bastos (UERJ),
Evandro L. C. Macedo (UERJ), Rafaela C. Brum (UERJ),
Alexandre Sztajnberg (UERJ), Diogo M. F. Mattos (UFF)

Abstract

This chapter presents the theoretical and practical foundations of security in Federated Learning (FL), focusing on major attacks that impact the performance and privacy of decentralized models. It examines vulnerabilities such as model poisoning, gradient inference, sample reconstruction with Generative Adversarial Networks (GAN), model inversion, and backdoor attacks, along with mitigation strategies like robust aggregation (median, trimmed mean, Krum), differential privacy, and homomorphic encryption. Participants will use the Flower framework to implement and evaluate attacks and defenses in simulated FL environments, applying reproducible code to assess the effectiveness of countermeasures.

Resumo

Este capítulo apresenta os fundamentos teóricos e práticos de segurança em Aprendizado Federado (Federated Learning - FL), com foco nos principais ataques que afetam o desempenho e a privacidade de modelos descentralizados. São examinadas vulnerabilidades como envenenamento de modelo, inferência de gradientes, reconstrução de amostras com Redes Adversariais Generativas (GAN), inversão de modelo e ataques backdoor, além de estratégias de mitigação como agregação robusta (median, trimmed mean, Krum), privacidade diferencial e criptografia homomórfica. Os participantes utilizarão o arcabouço Flower para implementar e avaliar ataques e defesas em ambientes simulados de FL, aplicando códigos reproduzíveis para avaliar a efetividade das contramedidas.

Este capítulo foi realizado com recursos do CNPq, CAPES - Código de Financiamento 001, RNP e FAPERJ. Ferramentas de Inteligência Artificial Generativa, incluindo ChatGPT, Grammarly e Perplexity, foram empregadas na revisão textual deste trabalho.

1.1. Introdução

Aprendizado Federado (*Federated Learning* – FL) é um paradigma de aprendizado de máquina decentralizado no qual múltiplos clientes treinam colaborativamente um modelo sem a necessidade de centralizar os dados brutos em um único repositório [Brendan McMahan et al., 2017]. Nesse paradigma, os dados permanecem localmente nos dispositivos ou nas organizações de origem. Cada cliente treina um modelo local com seus próprios dados e compartilha com um terceiro confiável, o servidor agregador, apenas atualizações de parâmetros, gradientes ou estatísticas agregadas [Lim et al., 2020]. Essa característica torna o Aprendizado Federado especialmente relevante em cenários nos quais privacidade, confidencialidade, soberania de dados e restrições regulatórias limitam a coleta centralizada de informações [Cunha Neto et al., 2023]. Aplicações embarcadas, sistemas móveis e serviços personalizados em larga escala têm demonstrado interesse pelo Aprendizado Federado como alternativa para treinamento descentralizado sem compartilhamento de dados dos usuários.

Iniciativas contínuas de pesquisa em Aprendizado Federado estão presentes na academia, literatura científica e têm sido conduzidas por grandes empresas de tecnologia. A Apple tem reportado arquiteturas de *private federated learning* para treinamento em dispositivos de borda e sistemas federados voltados à personalização e ao aprendizado em escala [Ji et al., 2025]. O Google, por sua vez, tem destacado implantações de Aprendizado Federado com garantias formais de privacidade diferencial em produção, inclusive em aplicações do Gboard [Hard et al., 2018], além de continuar expandindo técnicas de análise e adaptação federada em ambientes reais[1]. Em paralelo, a NVIDIA mantém o ecossistema NVIDIA FLARE como plataforma aberta para colaboração distribuída e treinamento federado em múltiplos domínios, incluindo aplicações recentes em dispositivos móveis e modelos de maior porte[2]. Em conjunto, essas iniciativas indicam que o Aprendizado Federado continua sendo uma linha tecnológica estratégica, impulsionada por demandas concretas de treinamento sobre dados descentralizados e por desafios ainda em aberto relacionados a eficiência, robustez e privacidade. Arcabouços como PySyft[3], Flower[4] e TensorFlow Federated (TFF)[5] têm desempenhado papel fundamental na viabilização do Aprendizado Federado.

Apesar desses benefícios, o Aprendizado Federado não é intrinsecamente seguro [Lyu et al., 2020]. A sua característica descentralizada e a consequente ausência de controle centralizado absoluto ampliam sua superfície de ataque, uma vez que os dispositivos clientes, agora incorporados ao processo de treinamento, podem introduzir vulnerabilidades adicionais [Cunha Neto et al., 2023]. Nesse contexto, os ataques podem ter finalidades distintas: busca por extrair informações sensíveis, como dados de treinamento, gradientes ou propriedades estatísticas dos conjuntos locais, ou a tentativa de comprometimento direto do processo de aprendizado, degradando a convergência, reduzindo a acurácia do modelo global ou inserindo comportamentos maliciosos específicos [Liu et al.,

[1]Disponível em https://research.google/blog/federated-learning-with-formal-differential-privacy-guarantees/. Acessado em 21/04/2026

[2]Disponível em https://developer.nvidia.com/flare. Acessado em 24/04/2026

[3]Disponível em https://docs.openmined.org/en/latest/. Acessado em 21/04/2026

[4]Disponível em https://flower.ai/. Acessado em 21/04/2026

[5]Disponível em https://www.tensorflow.org/federated?hl=pt-br. Acessado em 21/04/2026

2022]. Assim, surgem diferentes classes de ataque ao FL, incluindo ataques de envenenamento de modelo (*model poisoning*), envenenamento de dados, ataques de inversão de modelo, reconstrução baseada em redes adversárias generativas (*Generative Adversarial Networks* – GAN), ataques bizantinos e ataques *backdoor*. Em conjunto, esses ataques podem comprometer tanto a confidencialidade dos dados dos clientes quanto a integridade e o desempenho do modelo global [Liu et al., 2022]. Assim, em contextos sensíveis como saúde, finanças, cidades inteligentes e Internet das Coisas (*Internet of Things* – IoT), a proteção dos dados torna-se ainda mais importante, pois a exposição ou vazamento de dados e informações pode comprometer a segurança dos usuários, violar legislações de privacidade cada vez mais rigorosas e até mesmo atentar contra vidas [Bochie et al., 2021].

Este capítulo apresenta uma abordagem teórico-prática, propondo a implementação e análise experimental de diferentes estratégias de ataques. São conduzidos experimentos em um ambiente de Aprendizado Federado baseado no arcabouço Flower[6]. A fim de motivar os participantes e investigar a viabilidade de inferência de dados a partir de informações compartilhadas durante o treinamento, os participantes realizarão a implementação dos seguintes ataques:

- ***Deep Leakage from Gradients* (DLG):** neste ataque há a reconstrução de dados privados de um cliente ao ajustar entradas fictícias até que seus gradientes coincidam com os gradientes compartilhados no Aprendizado Federado.
- **Reconstrução com GAN:** nesse ataque, o atacante utiliza o modelo global como discriminador para treinar um gerador capaz de sintetizar imagens com características semelhantes às amostras presentes no conjunto de dados dos demais participantes, em especial da classe alvo que se deseja inferir.
- **Bizantinos:** neste ataque, clientes maliciosos enviam atualizações aleatórias ou manipuladas ao servidor de agregação, com o objetivo de degradar o desempenho do modelo global. Nessa etapa, serão implementados três ataques bizantinos, ruído gaussiano, *A Little is Enough* (ALIE) e inversão de sinal.

O restante do capítulo está organizado da seguinte forma. A Seção 1.2 apresenta a fundamentação teórica sobre o funcionamento do Aprendizado Federado. A Seção 1.3 discute os principais ataques, enquanto a Seção 1.4 expõe as estratégias de mitigação aos ataques. A atividade prática é detalhada na Seção 1.5 e a Seção 1.6 analisa as tendências de pesquisa e desafios futuros. A Seção 1.7 conclui o capítulo.

1.2. Fundamentação Teórica de Aprendizado Federado

O Aprendizado Federado é um paradigma de treinamento colaborativo em que um modelo global é aprendido a partir de dados mantidos em dispositivos ou instituições distribuídas, sem a necessidade de centralizar os dados brutos. Nesse processo, cada participante treina localmente uma cópia do modelo com seus próprios dados e transmite

[6]Disponível em https://flower.ai/. Acessado em 22/04/2026.

apenas as atualizações de parâmetros ao servidor agregador, preservando a confidencialidade dos conjuntos de dados. A cada rodada, o servidor coleta essas contribuições, combina as atualizações recebidas e produz uma nova versão do modelo global, que é redistribuída aos participantes para continuidade do treinamento, mostrado na Figura 1.1.

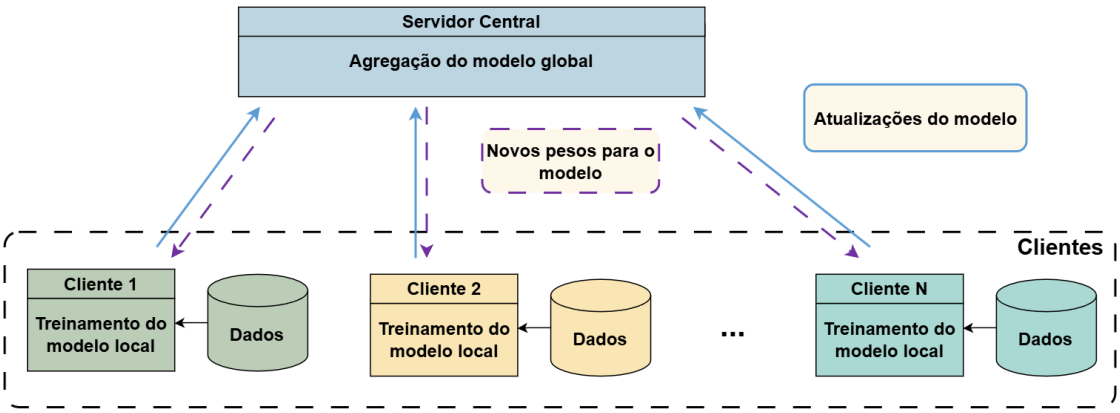


Figura 1.1. Fluxo geral do Aprendizado Federado em arquitetura centralizada. (i) O servidor envia os parâmetros atuais do modelo global aos clientes. (ii) Cada cliente realiza o treinamento local sobre seus próprios dados e retorna apenas atualizações do modelo. (iii) O servidor agrega essas contribuições para gerar um novo modelo global, sem a centralização de dados brutos dos participantes.

A geração do modelo global no Aprendizado Federado ocorre por meio de rodadas de comunicação em que se alternam treinamento local e agregação de parâmetros. Em cada rodada, o processo pode ser organizado em três etapas: (i) o servidor central seleciona os clientes participantes, que podem corresponder a uma fração ou à totalidade dos clientes, e envia a eles os parâmetros atuais do modelo; (ii) cada cliente escolhido atualiza seu modelo local com esses parâmetros, executa o treinamento por algumas épocas com seus dados privados e devolve ao servidor os parâmetros atualizados; (iii) ao receber as atualizações de todos os clientes participantes, o servidor agrega os parâmetros utilizando algoritmos específicos, como o FedAvg [McMahan et al., 2016], obtendo o modelo global daquela rodada de comunicação.

A dinâmica, mostrada na Figura 1.1, caracteriza a arquitetura de Aprendizado Federado centralizado e síncrono, na qual um servidor central coordena a agregação das atualizações dos clientes aguardando o recebimento de todos os parâmetros esperados [Zhang et al., 2025, Pang et al., 2025]. Nesse cenário, o servidor atua como ponto de coordenação e de decisão, o que simplifica o controle do processo de treinamento, mas também cria um ponto único de falha e potencial gargalo de comunicação. A partir dessa formulação centralizada e síncrona, variações da dinâmica original de Aprendizado Federado levam a arquiteturas alternativas, como o Aprendizado Federado descentralizado [Zhang et al., 2025] e o Aprendizado Federado assíncrono ou semi-assíncrono [Pang et al., 2025], que buscam mitigar limitações de robustez e escalabilidade.

O Aprendizado Federado descentralizado [Zhang et al., 2025] remove o papel do servidor central de agregação e distribui a responsabilidade de combinação dos modelos entre os próprios clientes (Figura 1.2). Nessa arquitetura, todos os clientes participam de todas as rodadas de comunicação e, ao final de seus treinamentos locais, enviam seus pa-

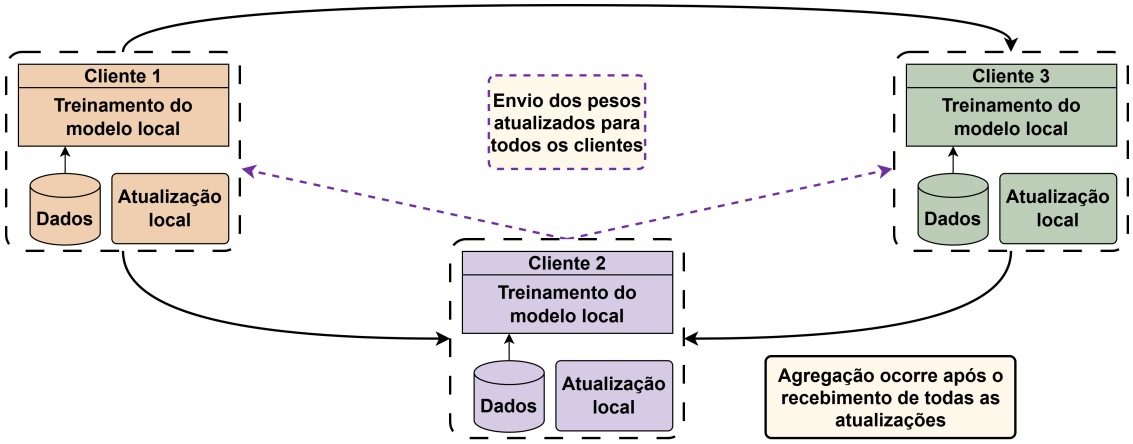


Figura 1.2. Arquitetura do Aprendizado Federado Descentralizado. Não há servidor central: cada cliente treina localmente, compartilha atualizações com seus vizinhos e realiza agregação distribuída. Essa abordagem elimina o ponto único de falha, ao custo de maior complexidade de comunicação e coordenação.

râmetros atualizados diretamente aos demais clientes, que então realizam uma agregação local, em geral utilizando algoritmos simples, como o gradiente descendente estocástico [Zhang et al., 2025]. Essa abordagem elimina o ponto único de falha associado ao servidor central, mas aumenta significativamente o uso da rede, pois o número de mensagens trocadas cresce rapidamente com o número de clientes, impondo novos desafios de escalabilidade e coordenação.

Duas outras variações relevantes, o Aprendizado Federado assíncrono e o semi-assíncrono, podem ser construídas tanto sobre a arquitetura centralizada quanto sobre a descentralizada [Pang et al., 2025]. Nesses casos, a principal diferença em relação ao paradigma síncrono aparece na etapa de agregação, representada pela etapa (iii) da descrição, modificando o momento e a forma como as atualizações dos clientes são incorporadas ao modelo global. Assim, a definição de como e quando agregar as contribuições de cada cliente permite controlar o compromisso entre o quão novas são as atualizações, a estabilidade do treinamento e o custo de comunicação.

No Aprendizado Federado síncrono, a agregação dos parâmetros é realizada apenas depois que o servidor central, ou o conjunto de clientes no caso descentralizado, recebe as atualizações de todos os clientes participantes da rodada. Já no Aprendizado Federado assíncrono, a agregação é executada imediatamente a cada chegada de novos parâmetros, o que tende a aumentar o número de mensagens enviadas pelo servidor para os clientes com versões atualizadas do modelo e pode provocar congestionamento na rede [Pang et al., 2025]. Esse comportamento tende a reduzir o impacto das atualizações de clientes mais lentos, que frequentemente treinam sobre versões defasadas do modelo global, afetando a convergência e a qualidade final do modelo [Pang et al., 2025]. Diante desses efeitos, surge o Aprendizado Federado semi-assíncrono, no qual o servidor (ou cada cliente) realiza a agregação apenas a cada x atualizações recebidas, com x menor que o número total de clientes, ou após um intervalo de m segundos, buscando equilibrar a frequência de atualização e a robustez frente a clientes lentos.

O Aprendizado Federado semi-assíncrono procura reduzir o excesso de atualizações de parâmetros observado no regime totalmente assíncrono, sem introduzir os atrasos

característicos do regime síncrono causados por poucos clientes lentos. Ao limitar a frequência de agregação por número de atualizações ou por janela de tempo, essa abordagem diminui o volume de mensagens e suaviza a variabilidade entre as versões de modelo vistas pelos clientes, preservando parte da eficiência comunicacional do assíncrono.

Independentemente da arquitetura adotada, o Aprendizado Federado pode ser classificado em dois tipos principais, a partir da distribuição das amostras e dos atributos entre os clientes [Yang et al., 2019]. Quando os clientes compartilham o mesmo conjunto de atributos para suas amostras, o cenário é caracterizado como Aprendizado Federado Horizontal; por outro lado, quando os cada cliente observa um subconjunto distinto de atributos das mesmas ou de diferentes amostras, o cenário é definido como Aprendizado Federado Vertical [Yang et al., 2019]. Essa distinção conceitual orienta tanto o desenho dos protocolos de treinamento quanto os mecanismos de privacidade e de comunicação adequados a cada contexto.

No contexto específico do Aprendizado Federado Horizontal, o sistema pode ser ainda categorizado como Aprendizado Federado entre Dispositivos (*Cross-Device Federated Learning*) ou entre Silos de Dados (*Cross-Silo Federated Learning*), de acordo com o tipo de cliente envolvido [Kairouz e McMahan, 2021]. No primeiro caso, os clientes são tipicamente dispositivos de menor capacidade, como celulares e sensores IoT, sujeitos a conectividade de rede instável e recursos computacionais limitados, o que exige estratégias de participação parcial e tolerância a falhas. No segundo caso, os clientes correspondem, em geral, a organizações ou servidores com maior poder computacional e conectividade mais estável, permitindo rodadas de comunicação mais robustas e com maior volume de dados.

1.3. Ataques em Aprendizado Federado

O Aprendizado Federado (FL) está sujeito a diversas ações externas que podem comprometer tanto o desempenho do modelo global quanto a privacidade dos dados dos clientes, usualmente caracterizadas como ataques [Cunha Neto et al., 2023]. Esses ataques são, em geral, agrupados em duas categorias principais: (i) aqueles que visam degradar ou manipular o desempenho do modelo e (ii) aqueles cujo objetivo é violar ou inferir informações sobre os dados privados dos clientes.

A Figura 1.3 mostra em quais etapas do processo federado podem ocorrer os principais vetores de ataque, bem como os mecanismos de defesa associados a essas etapas. Ataques de interceptação podem ocorrer no canal de comunicação, permitindo que um adversário observe atualizações trocadas entre clientes e servidor e, a partir delas, realize inferências sobre informações dos dados locais. No nível dos dados, o envenenamento de dados consiste na manipulação do conjunto utilizado no treinamento local, enquanto o envenenamento do modelo corresponde à alteração direta das atualizações enviadas ao servidor, com o objetivo de influenciar o modelo global. Como contramedidas, a privacidade diferencial e a criptografia homomórfica atuam sobre a informação compartilhada durante a comunicação, reduzindo o risco de inferência a partir de atualizações interceptadas ou observadas. Já a agregação robusta atua no servidor, buscando limitar o impacto de atualizações maliciosas no processo de treinamento.

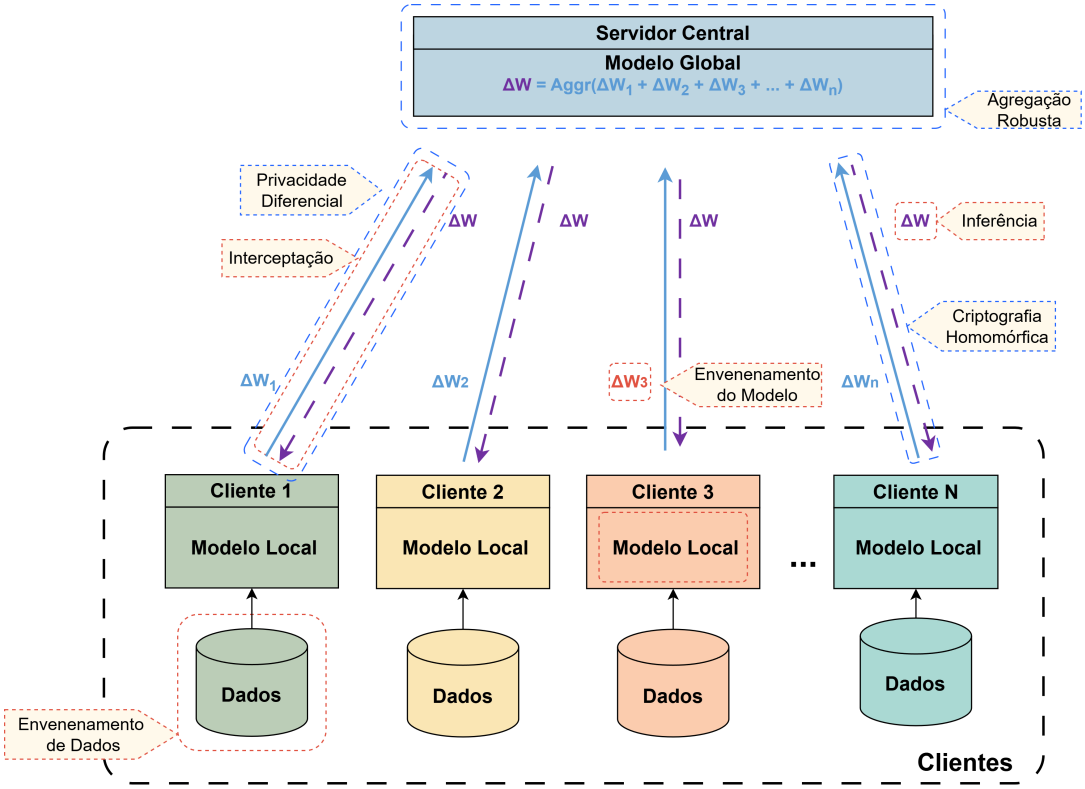


Figura 1.3. Superfícies de ataque e contramedidas no Aprendizado Federado. A figura mostra o fluxo entre servidor, clientes e dados locais, destacando ataques no canal de comunicação, no treinamento local e nas atualizações enviadas ao servidor. Também são indicadas defesas associadas a essas etapas, como privacidade diferencial, criptografia homomórfica e agregação robusta.

1.3.1. Ataques ao Desempenho do Modelo

O desempenho do modelo global em uma abordagem de Aprendizado Federado pode ser comprometido por clientes maliciosos, pois o servidor de agregação observa apenas as atualizações enviadas após o processamento local, sem visibilidade direta sobre o funcionamento interno que as gerou [Lim et al., 2020]. Nesse contexto, a literatura mostra que o modelo pode ser corrompido tanto por meio da adulteração do conjunto de dados utilizado no treinamento local [Goldblum et al., 2022] quanto pela manipulação explícita de gradientes ou parâmetros dos modelos dos clientes [Liu et al., 2022]. O desempenho global também pode ser afetado por ataques com finalidades distintas, como ataques bizantinos, voltados à degradação do treinamento, e ataques *backdoor*, voltados à inserção de comportamentos maliciosos específicos no modelo resultante.

Como essas ameaças diferem também na forma de execução, nesta seção as discussões são organizadas em três dimensões complementares. Em primeiro lugar, consideram-se as características operacionais do ataque, incluindo comportamento adaptativo, furtividade, operação descentralizada e execução assíncrona. Em seguida, analisa-se a finalidade do ataque, distinguindo-se ataques bizantinos de ataques *backdoor*. Por fim, avalia-se a superfície de manipulação, separando envenenamento do conjunto de dados e envenenamento do modelo; essa taxonomia, Figura 1.4, permite relacionar o objetivo do atacante, o ponto de interferência e as propriedades operacionais que determinam o impacto e a dificuldade de detecção de cada ataque.

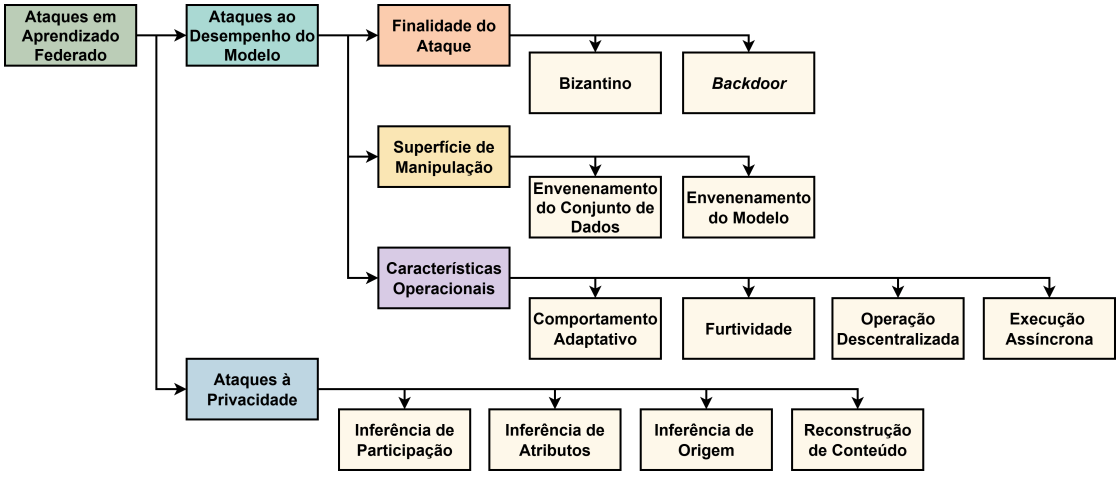


Figura 1.4. Taxonomia dos ataques em Aprendizado Federado. (i) Os ataques ao desempenho do modelo incluem ataques Bizantinos e backdoor, podendo atuar por envenenamento de dados ou do modelo e apresentando diferentes características operacionais, como adaptatividade, furtividade e execução distribuída. (ii) Os ataques à privacidade abrangem técnicas de inferência (participação, atributos e origem) e reconstrução de conteúdo.

Na dimensão de **Características Operacionais**, os ataques são analisados a partir de como são efetivamente executados no ambiente federado, considerando aspectos dinâmicos e estruturais de sua operação. Em particular, ataques com a mesma finalidade e a mesma superfície de manipulação podem diferir quanto ao nível de adaptação aos mecanismos de defesa, ao grau de furtividade buscado, à topologia do sistema em que são formulados e ao regime de sincronização adotado.

O *comportamento adaptativo* caracteriza ataques cuja construção depende explicitamente das propriedades do sistema federado alvo, em vez de se basear em uma perturbação fixa e independente do contexto. Nesses cenários, o adversário ajusta sua estratégia em função de elementos como a regra de agregação, os critérios estatísticos monitorados pela defesa, a dinâmica das rodadas de treinamento e, quando disponível, informações benignas de referência [Shejwalkar e Houmansadr, 2021]. A efetividade do ataque deixa de depender apenas da intensidade da perturbação e passa a depender também da capacidade de explorar as próprias premissas de robustez do sistema federado [Li et al., 2023].

A adaptatividade dos ataques pode se manifestar em diferentes níveis, a começar pela própria formulação da atualização maliciosa frente ao mecanismo de agregação robusta empregado pelo servidor. Em uma primeira forma de adaptação, o atacante constrói a perturbação já levando em conta quais padrões tendem a ser rejeitados pela regra de agregação e quais regiões do espaço de parâmetros permanecem aceitas, moldando o ataque para permanecer dentro dessas regiões [Li et al., 2023]. Em outra forma, a adaptação ocorre em relação às estatísticas observadas pelo sistema de detecção, de modo que a atualização maliciosa é ajustada para se manter compatível com métricas como médias, variâncias ou correlações monitoradas como indicadores de anomalia [Yang et al., 2025]. Essas duas frentes de adaptação abrem espaço para ataques que exploram simultaneamente os limiares de robustez e as métricas de monitoramento.

Em vez de fixar previamente uma única regra de ataque, o adversário pode ainda aprender, rodada após rodada, quais ações produzem maior degradação acumulada do

modelo global, fazendo com que a adaptatividade surja ao longo do próprio processo de treinamento. Nessa formulação, a estratégia maliciosa é atualizada com base na resposta observada do sistema, como em abordagens baseadas em aprendizado por reforço, nas quais o atacante aprende uma política para selecionar dinamicamente a perturbação mais eficaz em cada estado do treinamento federado [Li et al., 2022a]. Em cenários semi-assíncronos, essa adaptação pode incluir também a dimensão temporal, combinando a construção da atualização maliciosa com a escolha do momento e da frequência de participação dos clientes comprometidos [Pang et al., 2025].

A *furtividade* caracteriza ataques projetados para produzir efeito adversarial sem gerar sinais facilmente distinguíveis do comportamento benigno. Em Aprendizado Federado, essa propriedade é particularmente relevante porque o servidor, em geral, não observa os dados locais e frequentemente decide apenas com base nas contribuições agregadas ou em estatísticas resumidas das atualizações recebidas [Yang et al., 2025]. Diferentemente do comportamento adaptativo, que enfatiza o ajuste da estratégia adversarial ao agregador, à defesa ou à dinâmica do treinamento, a furtividade enfatiza a redução da detectabilidade da contribuição maliciosa, mesmo depois que o ataque já está em execução [Lyu et al., 2022]; assim, um ataque pode ser adaptativo sem ser furtivo, quando gera perturbações claramente anômalas, ou pode ser furtivo sem grande adaptatividade, quando apenas preserva uma aparência legítima sem um ajuste sofisticado ao contexto.

Em cenários recentes, a furtividade também passa a incorporar uma dimensão temporal, principalmente em ambientes semi-assíncronos. Nesses casos, a variabilidade introduzida pela obsolescência natural das atualizações pode servir para encobrir o ataque, tornando mais difícil distinguir contribuições benignas defasadas de atualizações adversariais cuidadosamente construídas [Pang et al., 2025]. Essa interação entre atraso, heterogeneidade temporal e comportamento malicioso amplia o espaço de ataque e reforça a importância de mecanismos de detecção que levem em conta a evolução das contribuições ao longo das rodadas.

A furtividade, portanto, não se resume à redução da magnitude da perturbação, mas à capacidade de formular contribuições adversariais compatíveis com os mecanismos de observação e decisão disponíveis no sistema [Lyu et al., 2022]. Em consequência, ataques furtivos tendem a ser especialmente desafiadores, pois exploram a lacuna entre causar dano e ser detectado, preservando a aparência de legitimidade da contribuição maliciosa ao longo de várias rodadas de treinamento [Yang et al., 2025].

A *operação descentralizada* caracteriza ataques formulados para ambientes de Aprendizado Federado em que não há um servidor central confiável responsável pela agregação das contribuições locais. Nesses cenários, cada cliente passa a trocar informações diretamente com um conjunto de vizinhos e atualiza seu modelo a partir de interações distribuídas ao longo da topologia de comunicação [Zhang et al., 2025]. Como consequência, a superfície de ataque deixa de se concentrar em um único ponto e passa a ser distribuída pela rede, permitindo que contribuições maliciosas se propaguem localmente e afetem sucessivas etapas de atualização [Fang et al., 2024].

Essa mudança altera a lógica do ataque, pois o adversário deixa de formular uma única contribuição maliciosa voltada a influenciar diretamente um agregado global e passa a explorar a própria estrutura do grafo de comunicação. Nesse contexto, o atacante disse-

mina modelos locais manipulados aos vizinhos, induzindo desvios progressivos na dinâmica distribuída de otimização, uma vez que a atualização de cada cliente passa a depender tanto do treinamento em seus dados quanto dos modelos recebidos de seu conjunto de vizinhos. Se w_i^t denota o modelo do cliente i na rodada t e $\mathcal{N}_i^t$ representa seu conjunto de vizinhos, a atualização local passa, portanto, a refletir a interação entre o modelo treinado localmente e os modelos recebidos de $\mathcal{N}_i^t$, de modo que a inserção de um modelo malicioso em uma vizinhança estratégica pode contaminar múltiplas trajetórias locais de aprendizado mesmo na ausência de um agregador central [Fang et al., 2024].

Essa característica torna a topologia da rede um componente central do ataque, pois a efetividade da contribuição adversarial passa a depender da conectividade do cliente malicioso, da variação temporal do grafo de comunicação e da ocorrência de desconexões parciais entre vizinhos. Em particular, quando o grafo é dinâmico, a propagação do efeito adversarial deixa de ser uniforme e passa a depender do caminho percorrido pela atualização manipulada, de forma que o ataque atua tanto sobre o conteúdo do modelo adulterado quanto sobre a maneira como clientes maliciosos exploram suas interações com vizinhos para difundir essa informação pela rede [Fang et al., 2024].

Outro aspecto importante é que, em ambientes descentralizados, o afastamento de uma atualização local em relação ao comportamento médio nem sempre decorre de ação maliciosa, especialmente quando o sistema está sujeito a mudança de distribuição (*distributional shift*). Nessa situação, uma contribuição pode parecer anômala por refletir uma alteração legítima no padrão estatístico dos dados do cliente, e não necessariamente um ataque, tornando mais difícil distinguir atualizações benignas não estacionárias de modelos locais maliciosos [Zhang et al., 2025].

Na dimensão de **Finalidade do Ataque**, os ataques são organizados de acordo com o objetivo principal perseguido pelo adversário. Diferentes ataques podem produzir efeitos distintos sobre o processo de treinamento colaborativo, seja influenciando a dinâmica de convergência, seja alterando o comportamento do modelo treinado. Alguns ataques têm como objetivo comprometer o treinamento colaborativo como um todo, degradando a convergência ou a qualidade final do modelo global, enquanto outros buscam inserir comportamentos maliciosos específicos que são acionados apenas sob determinadas condições de entrada [Cunha Neto et al., 2023]. Entre as diferentes finalidades de ataques ao FL, podem-se citar os atauqes Bizantinos e os ataques de *backdoor*.

Ataques Bizantinos correspondem a uma ameaça substancial ao treinamento colaborativo em FL [Shi et al., 2022]. Nesse tipo de cenário, clientes maliciosos podem se desviar arbitrariamente do protocolo esperado e enviar atualizações inconsistentes, manipuladas ou totalmente adversariais ao processo de agregação, tornando a identificação dos agentes maliciosos uma tarefa complexa. Diferentemente de uma simples perturbação aleatória, esse tipo de ataque é concebido para prejudicar o processo de agregação e induzir atualizações discrepantes em relação ao objetivo global de otimização, degradando a acurácia do modelo global e, em casos extremos, comprometendo a própria convergência do processo iterativo [Li et al., 2023, Sattler et al., 2020].

Ataques Bizantinos são uma ameaça central em Aprendizado Federado porque o atacante não precisa necessariamente produzir uma atualização evidentemente anômala para causar dano. Assim, o problema deixa de ser apenas a remoção de valores extremos

e passa a incluir a identificação de atualizações que, embora compatíveis com a distribuição observada, foram construídas para enviesar gradualmente a trajetória de otimização. Trabalhos recentes [Li et al., 2023, Fang et al., 2020, Shi et al., 2022] mostram que diversos ataques são projetados para contornar defesas *Byzantine-robust* e indicam que a robustez efetiva depende não apenas do agregador, como da distribuição estatística dos dados entre os clientes, do número de atacantes e da capacidade adaptativa do adversário.

A furtividade pode ser alcançada por diferentes estratégias de construção da atualização maliciosa. Em abordagens mais simples, o atacante pode, por exemplo, inverter o sinal do gradiente ou ampliar artificialmente sua magnitude, produzindo uma atualização que empurra o modelo global na direção oposta àquela esperada no treinamento benigno [Shi et al., 2022]. Em ataques mais eficazes, porém, a atualização maliciosa é construída com base na estatística observada das contribuições benignas. No ataque *A Little Is Enough* (ALIE), por exemplo, clientes maliciosos imitam a dispersão estatística das atualizações benignas ao estimar, para cada componente do vetor de atualização local, a média μ_i e o desvio padrão δ_i e escolher o valor corrompido dentro do intervalo $(\mu_i - z^{\max}\delta_i, ; \mu_i + z^{\max}\delta_i)$, em que $z^{\max}$ controla o afastamento admissível em unidades de desvio-padrão [Li et al., 2023]. Outra técnica consiste em alinhar a atualização maliciosa na direção oposta à média benigna, por exemplo definindo a atualização adversarial como uma versão escalada de $-\bar{g}$, em que $\bar{g}$ é o gradiente benigno médio, o que torna negativo o produto interno entre a direção agregada e o gradiente benigno e impede a descida adequada da função de perda [Li et al., 2023].

Há também ataques formulados explicitamente como problemas de otimização. Nesses ataques, o adversário escolhe uma direção de perturbação, um vetor que especifica como a atualização maliciosa deve se desviar de uma atualização benigna de referência no espaço de parâmetros, e ajusta as contribuições comprometidas para maximizar a diferença entre os gradientes agregados sob ataque e os gradientes agregados em um cenário benigno [Shejwalkar e Houmansadr, 2021, Fang et al., 2020]. Uma forma simplificada de representar esse tipo de ataque é definir a atualização maliciosa como $g_m = g_{\text{ref}} + p$, em que g_{ref} é uma atualização com aparência benigna e p é um vetor de perturbação que introduz o desvio adversarial, sendo o objetivo maximizar $|\tilde{g}(p) - \bar{g}|$, em que $\tilde{g}(p)$ é o gradiente agregado sob ataque e $\bar{g}$ é o gradiente agregado benigno. Para preservar a furtividade, impõem-se restrições que impedem que g_m se afaste excessivamente do conjunto benigno, por exemplo por meio de limites de distância em relação ao centro das atualizações benignas ou de intervalos coordenados definidos a partir da média e do desvio padrão observados em cada parâmetro [Shejwalkar e Houmansadr, 2021, Fang et al., 2020].

Outro aspecto recorrente é que o ataque bizantino não se restringe a cenários com servidor agregador centralizado. Em arquiteturas descentralizadas, a superfície de ataque se amplia, pois uma atualização maliciosa pode se propagar localmente pela rede e influenciar sucessivas etapas de agregação sem a presença de um servidor confiável [Fang et al., 2024]. Nesses cenários, clientes maliciosos podem disseminar modelos locais manipulados para os vizinhos, comprometendo a convergência do sistema e o desempenho final dos modelos aprendidos.

Além dos ataques maliciosos, ambientes descentralizados também estão sujeitos a efeitos de mudança de distribuição (*distributional shift*), que geram ruptura entre a dis-

tribuição dos dados de treinamento e a dos dados observados posteriormente em operação [Zhang et al., 2025]. Nesses casos, a degradação do modelo não decorre de comportamento adversarial, mas de alterações reais no padrão estatístico dos dados, de modo que uma atualização local pode se afastar do comportamento médio tanto por refletir uma mudança legítima na distribuição quanto por ter sido manipulada. Essa sobreposição entre heterogeneidade estatística e comportamento adversarial é crítica em Aprendizado Federado descentralizado, pois dificulta a distinção entre atualizações benignas não-estacionárias e modelos locais maliciosos [Zhang et al., 2025, Fang et al., 2024].

Há ainda formulações mais gerais em que o Ataque Bizantino é modelado de forma probabilística. Nesses modelos, assume-se um adversário onisciente capaz de comprometer apenas uma fração limitada da informação disponível e, ainda assim, escolher a perturbação mais danosa sob uma restrição explícita de esforço [Wang et al., 2025]. Esse adversário é tratado como um agente que conhece o classificador utilizado e o fenômeno verdadeiro subjacente aos dados, mas pode corromper apenas uma fração α das amostras, configurando uma hipótese adversarial forte, mais adequada à análise de pior caso do que à descrição de cenários operacionais típicos [Wang et al., 2025]. Essa perspectiva mostra que a força do ataque pode ser analisada não só pelo número de clientes comprometidos, mas também pelo grau de controle exercido sobre a distribuição das contribuições maliciosas e pelo custo para induzir degradação mensurável no desempenho de classificação.

O *ataque backdoor*, em aprendizado de máquina, é um ataque direcionado em que o adversário faz o modelo manter bom desempenho na tarefa principal, mas induz saídas incorretas sempre que a entrada contém um gatilho específico. Um exemplo clássico é um pequeno padrão fixo inserido em imagens, como um quadrado branco em um canto, que aciona uma predição equivocada apenas quando presente. Assim, o modelo aparenta funcionar normalmente na maioria dos casos, mas incorpora um comportamento malicioso latente ativado sob condições específicas, e, em Aprendizado Federado, esse ataque não precisa degradar visivelmente o desempenho global para ser bem-sucedido, pois sua efetividade depende justamente de preservar a utilidade aparente do modelo na tarefa principal enquanto introduz uma falha controlada em um subconjunto de entradas [Lyu et al., 2022, Wang et al., 2020, Cao et al., 2021].

Uma forma clássica de concretizar o ataque em Aprendizado Federado consiste em aproximar o modelo global do modelo treinado pelo atacante, estratégia frequentemente descrita como substituição do modelo (*model replacement*). Em sistemas baseados em média ponderada, como o FedAvg [McMahan et al., 2016], o adversário explora o fato de que, em cada rodada, o servidor envia o modelo global corrente w^t aos clientes selecionados, de modo que um cliente malicioso i_0 conhece esse estado e não precisa saber exatamente as atualizações dos demais. Assumindo que clientes honestos retornam modelos próximos de w^t, o atacante pode reescalar sua atualização e enviá-la como

$$\frac{n_S}{n_{i_0}}(w_{i_0} - w^t) + w^t, \tag{1}$$

em que n_{i_0} é o tamanho do conjunto de dados local do atacante e $n_S = \sum_{i \in S} n_i$ é o total de dados dos clientes selecionados; como a atualização global em FedAvg é dada por

$$w^{t+1} = w^t + \sum_{i \in S} \frac{n_i}{n_S}(w_i - w^t), \tag{2}$$

se os clientes honestos estiverem próximos de convergência, com $w_i \approx w^t$ para $i \neq i_0$, suas contribuições praticamente se anulam e a atualização reescalada do atacante passa a dominar a agregação, resultando em $w^{t+1} \approx w_{i_0}$ [Wang et al., 2020]. Nessa situação, o comportamento indevido não surge de ruído acumulado, mas de uma atualização planejada, construída para substituir, total ou parcialmente, a trajetória de aprendizado benigno.

Trabalhos mais recentes mostram, porém, que o ataque pode ser implementado de forma mais sutil do que uma simples substituição agressiva do modelo. Ao invés de enviar uma atualização evidentemente discrepante, o atacante pode otimizar localmente seu modelo malicioso impondo restrições para que ele permaneça próximo do modelo global corrente a cada rodada, tornando mais difícil a detecção por agregadores robustos. No caso dos *backdoors* por casos raros (*edge-case backdoors*), essa estratégia é combinada com o uso de amostras pertencentes à cauda da distribuição de entrada, exemplos semanticamente válidos, mas pouco frequentes no conjunto de dados, como por exemplo imagens reais de aviões de uma companhia específica rotuladas incorretamente como "caminhão" em um problema de classificação de imagens [Wang et al., 2020]. Nessa configuração, o gatilho deixa de ser um padrão artificial evidente e passa a ser associado a entradas raras, que tendem a aparecer com baixa probabilidade durante o treinamento e a avaliação convencional, o que reduz a chance de que o comportamento malicioso seja identificado por inspeções empíricas rotineiras.

A literatura também indica que a dificuldade de defesa contra *backdoors* em Aprendizado Federado decorre de limitações estruturais do próprio paradigma [Lyu et al., 2022, Wang et al., 2020]. Como o servidor não observa dados locais e, em cenários com agregação segura, pode ter acesso apenas ao resultado agregado, sem inspecionar individualmente as contribuições dos clientes, muitas defesas tradicionais contra *backdoor* desenvolvidas para aprendizado centralizado tornam-se inaplicáveis. Ademais, em Aprendizado Federado uma defesa precisa ser robusta tanto a manipulações nos dados locais quanto a manipulações diretas nas atualizações de modelo, o que faz com que abordagens focadas principalmente em ataques bizantinos ou em *outliers* não ofereçam necessariamente proteção adequada a ataques direcionados [Lyu et al., 2022]. A robustez a *backdoors* está fortemente associada à robustez a exemplos adversariais. Adicionalmente, a detecção de *backdoors* em modelos treinados pode ser computacionalmente difícil em cenários gerais. Assim, esse cenário reforça que o desafio vai além de projetar filtros melhores para atualizações suspeitas e envolve lidar com uma forma de comprometimento que pode permanecer praticamente invisível, ao mesmo tempo em que preserva a utilidade aparente do modelo na maior parte das entradas [Wang et al., 2020].

Na dimensão de **Superfície de Manipulação**, os ataques são organizados de acordo com a superfície explorada pelo adversário. A manipulação pode incidir sobre os dados que alimentam o treinamento local ou sobre a própria contribuição gerada ao final desse treinamento, como gradientes e parâmetros do modelo. Diferenciar esses tipos de manipulação é importante pois permite separar ataques que alteram indiretamente o comportamento do modelo global, por meio da experiência de treinamento do cliente, daqueles que atuam diretamente sobre a informação enviada ao mecanismo de agregação.

No *envenenamento do conjunto de dados*, o atacante compromete o processo de treinamento a partir da manipulação do conjunto de dados local utilizado no treinamento,

ao invés de atuar diretamente sobre a regra de agregação ou sobre a atualização final enviada ao servidor [Nowroozi et al., 2025]. A manipulação ocorre, portanto, sobre os dados do cliente malicioso, que passam a ser usados em um processo de treinamento aparentemente legítimo, mas baseado em informações corrompidas. Como a atualização local é produzida a partir desses dados adulterados, o efeito adversarial é transferido indiretamente para o modelo global, o que torna essa superfície de ataque especialmente relevante em Aprendizado Federado [Lyu et al., 2022].

Esse tipo de ataque consiste em alterar a distribuição utilizada no treinamento local. Essa alteração pode ocorrer por diferentes mecanismos, como modificação de rótulos, perturbação de características, inserção de amostras sintéticas ou substituição parcial do conjunto original, mas em todos os casos o objetivo final permanece o de influenciar o modelo global. O impacto pode ser tanto na degradação da tarefa principal quanto na indução de comportamentos direcionados em condições de entrada [Wang et al., 2020].

Uma forma particularmente simples de envenenamento é a inversão de rótulos (*label flipping*). O atacante altera o rótulo de parte das amostras locais, mantendo as entradas originais, forçando o modelo a aprender associações inconsistentes entre representação e classe e produzindo gradientes que deixam de apontar para a minimização correta da função de perda [Lyu et al., 2022]. Em sua forma mais direta, o ataque não exige conhecimento profundo sobre a arquitetura ou a regra de agregação, bastando acesso ao conjunto local e à capacidade de modificar as anotações, embora sua efetividade dependa do volume de dados adulterados, da distribuição das classes e da capacidade do treinamento global de absorver o ruído introduzido.

Outra estratégia consiste em manipular características das amostras sem alterar necessariamente seus rótulos. No ataque por envenenamento de características (*feature poisoning*), o atacante modifica atributos considerados influentes para a decisão do modelo, de modo a deslocar a fronteira de decisão sem produzir degradação abrupta e facilmente perceptível no desempenho global. Nowroozi *et al.* mostram o caso em que essa manipulação é guiada por uma floresta aleatória com *permutation feature importance* [Nowroozi et al., 2025], técnica em que os valores de cada característica são embaralhados e o impacto dessa perturbação no desempenho é usado como medida de relevância. Assim, seja I_k a importância estimada da característica k, o atacante seleciona a coluna mais influente $k^\star = \arg\max_k I_k$ e concentra a manipulação nessa dimensão, normalizando seus valores e substituindo a característica, em uma fração P das amostras maliciosas, por valores escolhidos aleatoriamente entre os valores únicos observados [Nowroozi et al., 2025]. Dessa forma, a perturbação incide sobre a variável que mais contribui para a decisão, aumentando a capacidade de deslocar a fronteira de classificação mesmo sem alterar todas as variáveis e mantendo métricas globais aparentemente elevadas [Nowroozi et al., 2025].

Há também ataques em que o envenenamento inclui a geração de novas amostras artificiais. No ataque baseado em Rede Generativa Adversarial (*Generative Adversarial Network* – GAN), o cliente malicioso começa atuando como um participante aparentemente benigno e, ao longo do treinamento federado, utiliza as informações trocadas para treinar localmente uma GAN que aprenda a imitar amostras prototípicas de classes que não estavam originalmente em sua posse [Zhang et al., 2019]. Em seguida, o atacante rotula essas amostras geradas de acordo com seu objetivo malicioso e as insere no trei-

namento local, alterando a composição semântica de seu conjunto de dados e produzindo atualizações que comprometem o modelo global, embora o processo continue se apresentando como um treinamento padrão baseado em dados.

No contexto de ataques direcionados, como *backdoors*, o envenenamento do conjunto de dados costuma ocorrer por inserção parcial de amostras maliciosas no conjunto local. Em vez de substituir completamente a base de treinamento, o atacante preserva exemplos limpos e adiciona amostras contendo o padrão de ativação do ataque, associadas a um rótulo alvo específico, caracterizando o chamado caso *black-box* de inserção de *backdoor* [Wang et al., 2020]. Nessa configuração, o adversário atua apenas sobre os dados, sem uma etapa adicional de otimização explícita da atualização enviada ao servidor: o comportamento malicioso é aprendido pelo modelo local a partir das amostras envenenadas e propagado ao modelo global por meio das atualizações resultantes, de modo que, mesmo quando a finalidade é um *backdoor*, a superfície de manipulação permanece o conjunto de dados [Wang et al., 2020].

Do ponto de vista operacional, o atacante, em envenenamento de dados, depende fortemente da dinâmica do treinamento local para transferir o efeito adversarial ao modelo global. Em alguns cenários, essa dependência torna o ataque menos potente do que abordagens que manipulam diretamente a atualização final enviada ao servidor [Nowroozi et al., 2025, Cunha Neto et al., 2023], pois parte do efeito adversarial pode ser absorvida pelo processo de otimização e pela agregação entre múltiplos clientes. Por outro lado, essa mesma característica tende a torná-lo mais natural e menos suspeito, já que a contribuição local é produzida por uma rotina de treinamento que, do ponto de vista do servidor, se assemelha ao comportamento regular de um cliente benigno [Lyu et al., 2022].

No *envenenamento do modelo*, a manipulação adversarial incide diretamente sobre a atualização que o cliente envia ao servidor agregador, em vez de atuar apenas sobre os dados usados no treinamento local [Lyu et al., 2022]. Em vez de depender do efeito indireto de dados corrompidos, o cliente malicioso constrói explicitamente um modelo adulterado, denotado por w_{i_0}, e o utiliza como vetor adversarial, de modo que o desvio introduzido passa a ser formulado diretamente no objeto matemático agregado pelo servidor [Shi et al., 2022].

Uma forma simples de implementar esse ataque consiste em inverter o sentido da atualização honesta ou ampliar artificialmente sua magnitude. Se a contribuição local legítima do cliente malicioso é $w_{i_0} - w^t$, o atacante pode substituí-la por

$$-\lambda(w_{i_0} - w^t), \qquad \lambda > 0, \tag{3}$$

onde λ é um fator de escala positivo que controla a intensidade do ataque [Shejwalkar e Houmansadr, 2021]. Esse princípio aparece em ataques de inversão de sinal (*sign flipping*) e também em variantes nas quais a magnitude da atualização maliciosa é ampliada para aumentar sua influência sobre a média agregada, induzindo o modelo global a evoluir na direção oposta àquela esperada no treinamento benigno [Shi et al., 2022].

Uma estratégia mais direcionada consiste em manipular explicitamente o alinhamento entre a atualização agregada e a direção benigna de otimização [Li et al., 2023]. Seja $\bar{u}$ a média das atualizações honestas, ou uma estimativa dessa média obtida a partir de contribuições benignas conhecidas pelo atacante, a contribuição maliciosa pode ser

construída na direção oposta, por exemplo $\tilde{u}_{i_0} = -\lambda \bar{u}$, com $\lambda > 0$ [Shejwalkar e Houmansadr, 2021]. Diferentemente da simples inversão da própria atualização local, essa construção visa interferir diretamente na direção média da agregação, buscando tornar negativo ou suficientemente pequeno, o produto interno entre a atualização agregada e a direção benigna e, assim, reduzir o alinhamento do treinamento com a descida da função de perda [Li et al., 2023].

Há também ataques formulados como problemas explícitos de otimização. Nesses casos, o adversário parte de uma atualização de referência u_{ref} e define uma direção de perturbação p, obtendo uma contribuição comprometida da forma

$$\tilde{u}_{i_0} = u_{\text{ref}} + p. \tag{4}$$

O vetor p é então escolhido para maximizar a diferença entre o agregado sob ataque $\tilde{u}_{\text{agg}}(p)$ e o agregado benigno u_{agg}, sob restrições que impeçam que a atualização maliciosa se afaste excessivamente do conjunto benigno [Li et al., 2023]. Em termos genéricos, pode-se escrever

$$\max_{p} \ \left\| \tilde{u}_{\text{agg}}(p) - u_{\text{agg}} \right\| \quad \text{sujeito a} \quad u_{\text{ref}} + p \in \mathscr{C} \tag{5}$$

em que $\mathscr{C}$ representa o conjunto de restrições impostas para reduzir a chance de rejeição pelo agregador, como limites de distância em relação ao centro das atualizações benignas [Shejwalkar e Houmansadr, 2021]. Essa lógica está no núcleo de duas famílias de ataques: os ataques adaptados à regra de agregação (*AGR-tailored*), em que o adversário conhece o mecanismo robusto empregado e o contorna explicitamente, e os agnósticos (*AGR-agnostic*), em que o agregador é desconhecido e a contribuição maliciosa é formulada de modo mais geral, mantendo-se próxima do conjunto benigno [Li et al., 2023].

Outra formulação é o desvio direcionado (*directed deviation*). Nesse caso, o atacante estima a direção s em que o modelo global evoluiria na ausência de ataque e constrói a contribuição comprometida para deslocar cada parâmetro no sentido oposto, como

$$\tilde{u}_{i_0} = u_{i_0} - \gamma s, \tag{6}$$

com $\gamma > 0$ controlando a intensidade do desvio [Fang et al., 2020]. O objetivo não é apenas gerar uma atualização arbitrária, mas interferir na trajetória de otimização de forma coordenada, componente a componente, aumentando a efetividade do ataque contra agregadores que filtram apenas *outliers* grosseiros [Fang et al., 2020].

Modelos ainda mais sofisticados combinam esse objetivo com restrições explícitas de furtividade. Uma estratégia é resolver localmente um problema de otimização multicritério, no qual o atacante minimiza simultaneamente a perda da tarefa maliciosa e um termo de proximidade em relação ao modelo global corrente, por exemplo

$$\min_{w_{i_0}} \ \mathscr{L}_{\text{adv}}(w_{i_0}) + \beta \| w_{i_0} - w^t \|^2, \tag{7}$$

em que $\mathscr{L}_{\text{adv}}$ representa a perda associada ao comportamento adversarial desejado e o termo $\beta \| w_{i_0} - w^t \|^2$ penaliza afastamentos excessivos do estado global [Wang et al., 2020]. Essa formulação permite preservar o efeito do ataque ao mesmo tempo em que

reduz a discrepância entre a contribuição maliciosa e o modelo benigno esperado [Wang et al., 2020].

O envenenamento do modelo também pode ser visto como um problema sequencial de decisão. Em vez de adotar uma heurística fixa, o adversário pode usar aprendizado por reforço (*Reinforcement Learning* – RL) para aprender, ao longo das rodadas, quais ações produzem maior degradação acumulada do modelo global [Li et al., 2022a]. Nesse contexto, o processo de ataque é modelado como uma sequência de interações em que, a cada rodada t, o atacante observa um estado s_t que resume informações relevantes sobre o treinamento, escolhe uma ação a_t conforme uma política parametrizada $a_t \sim \pi_\theta(s_t)$ e recebe um retorno que mede o impacto adversarial obtido [Li et al., 2022a]. O objetivo passa a ser aprender os parâmetros θ de uma política que selecione, em cada rodada, a contribuição maliciosa de maior impacto esperado, inclusive na presença de agregadores robustos [Li et al., 2022a].

O envenenamento do modelo não se restringe ao cenário centralizado síncrono. Em Aprendizado Federado descentralizado, a contribuição maliciosa circula entre vizinhos, permitindo que o ataque explore a topologia de comunicação para propagar atualizações manipuladas [Fang et al., 2024]. Em regimes semi-assíncronos, a defasagem temporal entre atualizações cria oportunidades adicionais, pois clientes distintos treinam sobre versões diferentes do modelo global e suas contribuições chegam ao servidor com diferentes níveis de obsolescência [Pang et al., 2025]. Nesse cenário, o adversário pode explorar não apenas o conteúdo da atualização maliciosa, mas também o momento e a frequência de participação dos clientes comprometidos, como no *framework* PoiSAFL, em que modelos locais envenenados são combinados com controle da frequência de envio para amplificação no processo semi-assíncrono de agregação [Pang et al., 2025].

1.3.2. Ataques à Privacidade

Os ataques à privacidade em Aprendizado Federado também podem ser categorizados pelo tipo de informação privada que o adversário busca extrair a partir do processo de treinamento. Dessa forma, os ataques são agrupados em quatro categorias: (i) ataques de inferência de participação, cujo objetivo é determinar se uma amostra específica integrou o conjunto de treinamento de um cliente [Shokri et al., 2017]; (ii) ataques de inferência de atributos, voltados à extração de propriedades latentes, sensíveis ou não intencionais presentes nos dados privados, mesmo quando não estão diretamente relacionadas à tarefa principal do modelo [Melis et al., 2019]; (iii) ataques de inferência de origem, nos quais o adversário busca associar informações, rótulos ou evidências a uma fonte específica, como um cliente do treinamento federado [Hu et al., 2023]; e (iv) ataques de reconstrução de conteúdo, cujo propósito é recuperar, aproximar ou sintetizar o próprio dado privado a partir de gradientes ou atualizações de modelo [Wu et al., 2024].

Os **ataques de inferência de participação** têm como objetivo decidir se uma amostra específica x pertenceu ou não ao conjunto de treinamento utilizado para ajustar um modelo alvo w. Formalmente, dado um modelo treinado w e uma amostra alvo x, o atacante busca estimar uma variável binária $m(x) \in 0,1$, em que $m(x) = 1$ indica que x foi usada no treinamento e $m(x) = 0$ indica o contrário [Shokri et al., 2017]. Em termos probabilísticos, o ataque consiste em construir um inferidor A capaz de aproximar

$P(m(x) = 1 \mid \psi(x,w))$, em que $\psi(\cdot)$ representa o conjunto de observáveis disponíveis ao adversário, como vetor de probabilidades, perda, ativações internas, gradientes locais ou atualizações federadas [Melis et al., 2019]. A relevância desse tipo de ataque decorre das evidências de participação já constituir uma violação severa de privacidade, mesmo sem revelar explicitamente o conteúdo do registro, em diversos domínios sensíveis.

Na formulação clássica [Shokri et al., 2017], considera-se o cenário de caixa preta (*black-box*) de acesso ao modelo, em que o adversário observa apenas a saída $w(x)$, normalmente expressa como um vetor de probabilidades sobre as classes. O ataque é convertido em um problema supervisionado auxiliar, em que se constrói um modelo atacante A que recebe como entrada a saída do modelo alvo e produz como saída a classe "membro" ou "não membro". Para obter dados rotulados para treinar A, utiliza-se a técnica de modelos-sombra (*shadow models*), na qual o atacante treina modelos auxiliares $w^{(1)}, \ldots, w^{(k)}$ em conjuntos de dados sob seu controle, de modo a reproduzir, de forma aproximada, o comportamento estatístico do modelo alvo. Para cada modelo auxiliar $w^{(i)}$, o atacante sabe exatamente quais amostras foram usadas no treinamento e quais foram mantidas fora, de modo que, ao consultar $w^{(i)}$ com exemplos desses dois grupos, obtém pares rotulados da forma $(w^{(i)}(x), m(x))$, que são então utilizados para treinar o classificador de ataque.

Melis *et al.* mostram que a inferência de participação não se limita a modelos centralizados acessados via *Application Programming Interface* (API), mas também surge naturalmente em aprendizado colaborativo [Melis et al., 2019]. A atualização compartilhada por cada cliente é calculada diretamente a partir de um lote local de dados e, portanto, preserva traços estatísticos desse lote: em uma rodada t, dado o modelo global w^t, cada cliente computa localmente um gradiente $\nabla L(b; w^t)$ sobre um lote b de seus dados e envia essa atualização ao servidor de agregação. Como direção e magnitude desse gradiente dependem explicitamente das amostras presentes em b, a atualização funciona como uma projeção do conteúdo do lote no espaço de parâmetros do modelo, o que permite a um cliente adversário, ao observar gradientes individuais ou diferenças sucessivas entre parâmetros globais, explorar essas variações para inferir se uma determinada amostra esteve presente no treinamento dos demais participantes [Melis et al., 2019].

O vazamento torna-se particularmente claro em camadas de entrada esparsas, como em *embeddings*, nas quais apenas as coordenadas associadas aos *tokens* efetivamente presentes no lote recebem gradientes não nulos, revelando diretamente a ocorrência desses elementos. Mesmo quando o protocolo de Aprendizado Federado expõe apenas o modelo global agregado, o adversário pode explorar a evolução temporal dos parâmetros, pois a diferença $\Delta w^t = w^t - w^{t-1}$ representa o efeito acumulado das atualizações dos participantes selecionados na rodada t. Se o adversário também participa do treinamento, sua própria contribuição local é conhecida e pode ser subtraída de Δw^t, produzindo uma estimativa da atualização agregada gerada pelos demais clientes, que ainda contém traços dos dados usados no treinamento local dos participantes honestos [Melis et al., 2019].

No cenário caixa branca (*white-box*), o adversário não se limita a observar a saída preditiva $w(x)$, mas explora parâmetros internos do modelo avaliados sobre a amostra alvo x. Esses parâmetros incluem ativações intermediárias das camadas ocultas, a perda $\mathscr{L}(w(x), y)$ e, principalmente, o gradiente dessa perda em relação aos parâmetros da rede,

$g(x) = \nabla \mathscr{L}(w(x), y)$. Entre essas quantidades, o vetor gradiente é o mais informativo para inferência de participação, pois expressa quanto e em que direção cada parâmetro ainda precisaria ser ajustado para reduzir a perda associada à amostra x; como o treinamento por gradiente descendente estocástico (*Stochastic Gradient Descent* – SGD) atualiza iterativamente os parâmetros para minimizar a perda sobre exemplos vistos, registros pertencentes ao conjunto de treinamento tendem a induzir padrões de gradiente estatisticamente distintos daqueles produzidos por amostras não-membros, de forma que cada exemplo de treino deixa uma "assinatura" no estado final dos parâmetros, mais evidente no espaço dos gradientes do que apenas na saída final do modelo [Zhu et al., 2025].

Com base nessa observação, Nasr *et al.* propõem ataques de inferência de participação em cenário caixa branca que utilizam gradientes locais por camada como entrada de um modelo atacante [Nasr et al., 2019]. Os autores mostram que mesmo modelos bem generalizados podem permanecer vulneráveis quando o adversário tem acesso aos parâmetros e às quantidades derivadas do processo de otimização. Por exemplo, para verificar se uma amostra x pertence ao conjunto de treinamento, o atacante pode, no cenário *white-box*, avaliar x no modelo treinado, calcular a perda $\mathscr{L}(w(x), y)$ e obter o gradiente local, que indica quanto cada parâmetro ainda precisaria ser ajustado para reduzir a perda associada a x [Xia et al., 2024]. Se x tiver sido usada no treinamento, espera-se que o modelo já esteja parcialmente adaptado a essa amostra, de modo que $g(x)$ apresente um padrão estatístico compatível com amostras membros. Caso contrário, o gradiente tende a refletir maior desalinhamento entre a amostra e o estado final dos parâmetros, sendo então utilizado, em suas componentes por camada, como entrada de um classificador de ataque que estima a probabilidade de x ser membro do conjunto de treinamento.

Os **ataques de inferência de atributos** têm como objetivo extrair propriedades dos conjuntos de dados dos clientes utilizados no treinamento do Aprendizado Federado [Melis et al., 2019]. Tais propriedades podem ser inferidas mesmo quando não constituem o alvo da tarefa principal aprendida pelo modelo global. Diferentemente da inferência de participação, em que o adversário busca decidir se uma amostra específica integrou o conjunto de treinamento, a inferência de atributos procura responder se um lote, subconjunto ou distribuição local de dados apresenta uma propriedade p de interesse [Diana et al., 2025]. Formalmente, considere uma amostra anotada como $z = (x, y, p)$, em que x representa a entrada, y o rótulo da tarefa principal e $p \in \{0, 1\}$ uma propriedade privada latente, potencialmente independente de y; enquanto y define o que o modelo foi treinado para prever, p define o que o atacante deseja descobrir a partir do comportamento do modelo ou das atualizações compartilhadas. Nessa formulação, o adversário busca estimar uma variável do tipo

$$a(b) = \mathbb{I}\{\exists z \in b : p(z) = 1\}, \tag{8}$$

ou, em variantes mais gerais, a fração de exemplos em um lote b que satisfazem p, de modo que o ataque não pretende reconstruir o dado bruto nem decidir sobre a presença de uma amostra específica, mas inferir uma característica semântica emergente do comportamento do modelo frente às atualizações induzidas pelos dados privados.

Enquanto abordagens centradas em classes exploram propriedades que caracterizam uma categoria inteira, a inferência de atributos [Melis et al., 2019] opera sobre propriedades válidas apenas para subconjuntos dos dados e potencialmente independentes da

tarefa alvo. Em vez de tentar inferir como é, em média, a classe y, o adversário procura inferir se os dados usados em uma atualização específica apresentam a propriedade p, o que é relevante porque o vazamento pode ocorrer mesmo quando essa propriedade não contribui para o desempenho preditivo do modelo, mas é capturada como consequência da riqueza estatística dos dados de entrada [Wang et al., 2022]. Embora o modelo seja treinado para a tarefa principal, as atualizações compartilhadas ainda podem carregar sinal suficiente para inferir atributos colaterais dos dados privados, de modo que o canal de vazamento deixa de ser o rótulo principal y e passa a ser a estrutura estatística das representações internas e dos gradientes associados a propriedades que o modelo não deveria, em princípio, revelar [Jayaraman e Evans, 2022].

No ataque passivo, o adversário não altera o protocolo de treinamento e utiliza apenas as atualizações compartilhadas ao longo das rodadas de agregação. Seja $\Delta\theta_t$ a atualização observada na rodada t, obtida diretamente a partir dos gradientes locais em protocolos descentralizados, nos quais ocorre troca explícita de gradientes, ou indiretamente como diferença entre instâncias consecutivas do modelo global; o ataque consiste em treinar um classificador auxiliar a partir de atualizações geradas por dados auxiliares rotulados segundo a propriedade privada p. Mais especificamente, o atacante utiliza um conjunto auxiliar do mesmo domínio, ou suficientemente próximo, para formar lotes com e sem a propriedade de interesse, obtendo pares $(\Delta\theta(b), a(b))$, em que $\Delta\theta(b)$ representa a atualização produzida pelo lote b e $a(b) \in \{0,1\}$ indica a presença ou ausência da propriedade nesse lote, que são então usados para treinar um classificador

$$A_{\text{prop}} : \Delta\theta \mapsto \{0,1\}, \tag{9}$$

capaz de estimar se a atualização observada no protocolo de Aprendizado Federado foi gerada por dados que contêm a propriedade privada [Melis et al., 2019].

Existe também uma variante ativa em que o participante adversário modifica seu próprio objetivo de treinamento para induzir o modelo conjunto a representar de forma mais nítida a propriedade privada de interesse [Melis et al., 2019]. A ideia é acoplar à tarefa principal uma tarefa auxiliar de predição de propriedade, substituindo a otimização puramente voltada à tarefa de classificação original por um objetivo multitarefa da forma

$$\mathscr{L}_{\text{adv}} = \mathscr{L}_{\text{main}} + \lambda\, \mathscr{L}_{\text{prop}}, \tag{10}$$

em que $\mathscr{L}_{\text{main}}$ corresponde à perda da tarefa legítima, $\mathscr{L}_{\text{prop}}$ força a rede a separar internamente exemplos que satisfazem ou não p, e $\lambda > 0$ controla a intensidade da interferência adversarial. Nesse cenário, o adversário não apenas observa o vazamento, mas amplifica a informatividade das atualizações futuras ao induzir o modelo global a aprender a propriedade p nas representações intermediárias, transformando o protocolo de Aprendizado Federado em um mecanismo de extração orientada de informação sem exigir acesso direto aos dados privados [Melis et al., 2019].

Os **ataques de inferência de origem** têm como objetivo associar uma informação privada a uma fonte específica dentro do Aprendizado Federado. Essa origem pode assumir diferentes formas, como o usuário responsável por um registro, o cliente detentor de um rótulo privado ou a pessoa à qual um padrão reconstruído pode ser atribuído [Fu et al., 2022]. Seja $\mathscr{C} = \{1, \ldots, K\}$ o conjunto de clientes participantes; dado um registro x,

um conjunto de características observáveis $\psi(x)$ e um mecanismo atacante A, o problema pode ser formulado como uma tarefa de atribuição

$$A(\psi(x)) \to \hat{c} \in \mathscr{C}. \tag{11}$$

Uma instância desse tipo de inferência aparece em Aprendizado Federado vertical, no qual diferentes participantes compartilham o espaço de amostras, mas não o espaço de atributos. Fu *et al.* analisam um cenário em que apenas um participante possui os rótulos y, enquanto os demais detêm subconjuntos de atributos $x^{(1)}, \ldots, x^{(K)}$ [Fu et al., 2022]; o objetivo do atacante deixa de ser decidir se uma amostra participou do treinamento ou reconstruir diretamente seu conteúdo, passando a ser a recuperação do rótulo privado mantido por outro participante. Assim, dada uma amostra x e o modelo federado treinado, o adversário busca estimar

$$\hat{y} = \arg\max_{y} P(y \mid x^{(a)}, \mathscr{I}), \tag{12}$$

em que $x^{(a)}$ representa os atributos observados pelo participante adversário e $\mathscr{I}$ agrega as informações disponíveis durante o treinamento, como gradientes recebidos e o comportamento do modelo de base (*bottom model*). No Aprendizado Federado vertical, cada participante utiliza um modelo de base para transformar seus atributos locais em uma representação intermediária, que é posteriormente combinada pelo modelo superior (*top model*), responsável por gerar a predição global, de modo que o modelo de base controlado pelo participante sem rótulos pode aprender representações suficientemente alinhadas com y para servir como instrumento de inferência [Fu et al., 2022].

Hu *et al.* apresentam uma formulação mais direta de inferência de origem [Hu et al., 2023]. Nesse caso, o objetivo não é apenas descobrir se um registro é membro do conjunto de treinamento global, mas inferir qual cliente possui esse registro, tratando a inferência de origem como um problema de atribuição entre os clientes da federação. O adversário observa, para um mesmo registro, como os diferentes modelos locais respondem em termos de erro preditivo e utiliza essa informação como critério de decisão, sob a hipótese de que o cliente que efetivamente utilizou esse registro em seu treinamento local tende a possuir um modelo mais ajustado à amostra, refletindo menor perda em comparação com os demais [Hu et al., 2023]. Essa ideia é adaptada a três protocolos de agregação distintos, *Federated Stochastic Gradient Descent* (FedSGD), FedAvg e *Federated Model Distillation* (FedMD), para demonstrar que a inferência de origem persiste independentemente de a federação compartilhar gradientes, parâmetros ou predições sobre dados públicos [Hu et al., 2023].

Os **ataques de reconstrução de conteúdo** têm como objetivo recuperar, aproximar ou sintetizar o próprio dado privado utilizado no treinamento. A reconstrução de conteúdo procura obter uma representação explícita da entrada privada, como uma imagem, um texto ou um protótipo semântico associado aos dados do participante. Seja $\mathscr{I}$ o conjunto de observáveis disponíveis ao adversário, como gradientes e parâmetros do modelo global, o objetivo do ataque pode ser escrito como a construção de um estimador $\hat{x} = A(\mathscr{I})$, tal que $\hat{x}$ preserve, na maior medida possível, a similaridade semântica ou estrutural com x. Em termos de otimização, isso equivale a buscar uma reconstrução

que minimize uma distância $d(\hat{x},x)$, ainda que x não seja diretamente observável pelo adversário [Zhu et al., 2019].

Zhu *et al.* formulam o problema e introduzem o ataque *Deep Leakage from Gradients* (DLG) [Zhu et al., 2019]. O cenário considerado é o de aprendizado distribuído ou colaborativo em que o adversário observa os gradientes compartilhados por outro cliente. Seja (x,y) um par privado de entrada e rótulo, $F(\cdot;W)$ um modelo diferenciável e o gradiente verdadeiro dado por

$$\nabla W = \frac{\partial \mathscr{L}(F(x;W),y)}{\partial W}, \tag{13}$$

o ataque parte de uma entrada fictícia x' e de um rótulo fictício y', ambos inicializados aleatoriamente, e computa os gradientes correspondentes

$$\nabla W' = \frac{\partial \mathscr{L}(F(x';W),y')}{\partial W}. \tag{14}$$

Em seguida, resolve-se o problema de otimização

$$\min_{x',y'} \left\| \nabla W' - \nabla W \right\|_2^2, \tag{15}$$

em que x' e y' são atualizados para minimizar a discrepância entre os gradientes fictícios e os gradientes observados.

A intuição é que, se os gradientes calculados sobre (x',y') coincidirem com os gradientes originalmente compartilhados, então o par fictício se torna uma aproximação do par privado real. O resultado central de DLG é mostrar que esse procedimento pode recuperar imagens com elevada fidelidade pixel a pixel e textos, evidenciando que o gradiente compartilhado preserva informação suficiente para reconstrução explícita do conteúdo [Zhu et al., 2019].

Apesar de sua importância, DLG apresenta limitações práticas, especialmente no tratamento do rótulo. Zhao *et al.* mostram que, no DLG original, o atacante precisa ajustar simultaneamente uma entrada fictícia x' e um rótulo fictício y' para que os gradientes gerados por esse par coincidam com os gradientes observados [Zhao et al., 2020a]. Como o rótulo verdadeiro não é previamente conhecido, ele também é tratado como variável de otimização, o que torna o problema mais instável e dificulta a convergência para a reconstrução correta. Para contornar esse problema, os autores propõem o *Improved Deep Leakage from Gradients* (iDLG), cuja principal contribuição é revelar que, em tarefas de classificação com perda de entropia cruzada e rótulos *one-hot*, o rótulo verdadeiro pode ser inferido analiticamente a partir dos sinais do gradiente da última camada totalmente conectada [Zhao et al., 2020a].

Seja g_i o gradiente da perda em relação ao *logit* da classe i, dado por $g_i = \frac{\partial \mathscr{L}}{\partial y_i}$. Os autores demonstram que, para a classe correta c, vale $g_c \in (-1,0)$, enquanto, para as demais classes $i \neq c$, $g_i \in (0,1)$. Quando os gradientes em relação aos *logits* não estão diretamente disponíveis, a mesma estrutura de sinal pode ser observada no gradiente dos parâmetros da última camada, permitindo identificar o rótulo verdadeiro antes da

reconstrução da entrada. Com isso, o problema de otimização deixa de envolver simultaneamente x' e y', reduzindo-se à

$$\min_{x'} \left\| \nabla W'(x', c) - \nabla W \right\|_2^2, \tag{16}$$

em que c já foi inferido. Essa modificação torna o processo mais estável, acelera a convergência e melhora a fidelidade da reconstrução, evidenciando que parte substancial do conteúdo supervisionado já vaza diretamente na estrutura do gradiente [Zhao et al., 2020a].

A reconstrução de conteúdo também pode ser conduzida por mecanismos generativos. Hitaj *et al.* mostram que, em aprendizado colaborativo profundo, um participante adversário pode utilizar Redes Generativas Adversariais (*Generative Adversarial Networks* – GANs) para sintetizar amostras prototípicas associadas aos dados privados de outro participante [Hitaj et al., 2017]. O ataque parte da observação de que, durante o treinamento, os parâmetros do modelo global são continuamente influenciados pelos dados locais dos participantes honestos. O adversário explora esse processo em tempo real, utilizando o modelo global como discriminador em uma GAN. Seja $G(z)$ o gerador, em que z é um vetor de ruído, e seja $D(\cdot)$ o discriminador fornecido pelo próprio treinamento colaborativo. O treinamento adversarial segue a lógica usual,

$$\min_G \max_D \; \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))], \tag{17}$$

mas, diferentemente do uso benigno de GANs para modelagem de distribuição, nesse caso o gerador é empregado de forma maliciosa para aproximar a distribuição dos dados privados do participante alvo [Hitaj et al., 2017]. É importante ressaltar que, quando a GAN é treinada, a esperança é aproximada pela média no *mini-batch*,

$$\mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] \approx \frac{1}{m} \sum_{i=1}^{m} \log D(x_i). \tag{18}$$

O ataque é particularmente relevante porque não visa necessariamente reconstruir uma amostra exata do conjunto local, como em DLG e iDLG, mas produzir conteúdo sintético suficientemente semelhante para revelar informação sensível sobre a distribuição privada aprendida durante o treinamento colaborativo [Hitaj et al., 2017].

Do ponto de vista operacional, os ataques DLG e iDLG são ataques de reconstrução por otimização guiada por gradientes. O objeto central observado pelo adversário é o gradiente compartilhado e a reconstrução decorre do ajuste iterativo de uma entrada fictícia até que ela induza o mesmo padrão diferencial do exemplo real [Zhu et al., 2019, Zhao et al., 2020a]. Já na reconstrução com GAN, o adversário utiliza a evolução do modelo colaborativo para treinar um gerador capaz de produzir amostras sintéticas da distribuição privada do alvo [Hitaj et al., 2017]. Em ambos os casos, a hipótese subjacente é a mesma de que o processo de treinamento colaborativo não transmite apenas conhecimento abstrato sobre a tarefa, mas preserva informação suficiente para aproximar o conteúdo privado que o originou.

1.4. Estratégias de Mitigação

O paradigma do Aprendizado Federado é ciente de privacidade para o treinamento de modelos, já que não há necessidade de compartilhar dados brutos e os participantes

podem decidir livremente ingressar ou deixar a federação, mas não fornece garantias suficientes de privacidade e robustez. Os mecanismos atuais tendem a serem vulneráveis, principalmente, (i) ao servidor curioso ou malicioso que busca inferir informações sensíveis dos clientes a partir das atualizações do modelo e (ii) a participantes adversários que têm como objetivo envenenar o modelo global [Lyu et al., 2022].

1.4.1. Técnicas de Preservação de Privacidade

Para ataques de inferência e de inversão de modelo, a privacidade diferencial surge como uma abordagem promissora. A privacidade diferencial define um arcabouço matemático que garante privacidade aos usuários por meio da inserção de ruído aleatório durante a análise ou o compartilhamento de dados. No contexto de Aprendizado Federado, destacam-se três tipos de soluções com privacidade diferencial: (i) a privacidade diferencial centralizada (*Centralized Differential Privacy* – CDP) [McMahan et al., 2018]; (ii) a privacidade diferencial local (*Local Differential Privacy* – LDP) [Wang et al., 2019, Zhao et al., 2020c, Truex et al., 2020, Sun et al., 2021]; e (iii) a privacidade diferencial distribuída (*Distributed Differential Privacy* – DDP) [Agarwal et al., 2018, Truex et al., 2019, Lyu, 2020]. Nessas variantes, a garantia de privacidade recai sobre valores agregados ou valores locais, a depender de quem é responsável por adicionar o ruído.

Os algoritmos CDP foram inicialmente descritos para aprendizado de máquina centralizado, no qual todos os dados se encontram em um único repositório. Ao adaptá-los ao Aprendizado Federado, assume-se que o servidor central é confiável e fica encarregado de adicionar o ruído aleatório após agregar os parâmetros recebidos dos clientes, utilizando, por exemplo, distribuições laplaciana ou gaussiana [Dwork e Roth, 2014]. Essa suposição de confiança, porém, nem sempre é realista e introduz um ponto único de falha para vazamentos de dados dos clientes, motivo pelo qual, em grande parte das aplicações de FL, recorrem-se a algoritmos baseados em LDP e DDP.

Nos algoritmos LDP, a responsabilidade por inserir ruído recai sobre os próprios clientes, que perturbam seus dados privados antes mesmo do treinamento local e do envio de parâmetros ao servidor. Para gerar esse ruído, o cliente pode empregar as mesmas distribuições laplaciana ou gaussiana usadas em CDP ou ainda modificar seus dados de modo que o servidor não consiga distinguir se os valores agregados refletem atualizações reais ou valores previamente determinados. Como o cliente desconhece o impacto exato de seus dados no modelo global, ele tende a inserir quantidades maiores de ruído para garantir privacidade diferencial, o que afeta a convergência do modelo, especialmente em cenários com centenas ou milhares de clientes, nos quais o erro aditivo em LDP é bem maior do que em CDP. Como consequência, muitas técnicas de LDP são mais adequadas a modelos com poucos parâmetros e a conjuntos de dados simples.

Como algoritmos LDP puros prejudicam significativamente a convergência e algoritmos CDP dependem de uma hipótese forte de confiança no servidor, surgiram os algoritmos DDP, que combinam inserção de ruído em parte dos dados dos clientes com funções criptográficas no servidor. Essa combinação restringe o conjunto de distribuições adequadas, privilegiando distribuições mais estáveis, como a gaussiana ou a binomial, que permitem a cada cliente adicionar ruídos de menor magnitude do que em LDP, ainda que a privacidade diferencial só seja estritamente satisfeita após a agregação dos valores

de todos os clientes [Agarwal et al., 2018]. Devido a essa característica, algoritmos DDP são tipicamente associados a mecanismos de computação multipartidária segura (*Secure Multi-Party Computation* – SMC), para reforçar as garantias de segurança sobre os dados dos participantes [Agarwal et al., 2018].

Outra abordagem é a criptografia homomórfica, técnica na qual computações podem ser realizadas diretamente sobre dados cifrados, sem necessidade de descriptografá-los [Lim et al., 2020, de Assis et al., 2023, de Assis et al., 2024]. Quando os dados são finalmente descriptografados, o resultado coincide com aquele que seria obtido ao realizar a mesma computação em texto plano, o que permite executar operações de agregação diretamente sobre dados cifrados e impede que um servidor não confiável acesse informações sensíveis e realize ataques de inferência. Embora ofereça um alto nível de proteção contra ataques de inferência por parte de servidores curiosos, o custo computacional dessa abordagem é elevado e pode ser inviável em dispositivos com recursos limitados, como aqueles comumente empregados em cenários de IoT.

1.4.2. Agregação Robusta

Técnicas de agregação robusta procuram combinar as atualizações de modelo dos participantes por meio de algoritmos resilientes a valores atípicos, ruído e manipulação adversária [Pillutla et al., 2022]. Em geral, esses métodos favorecem atualizações consistentes com a maioria e reduzem a influência de clientes maliciosos, sendo úteis contra ataques Bizantinos e ataques de envenenamento de modelo [Cunha Neto et al., 2023].

Ataques de envenenamento não-direcionados, em especial ataques Bizantinos, têm como objetivo comprometer a convergência do modelo global por meio da inserção de atualizações maliciosas. A defesa contra esses ataques é desafiadora porque o servidor observa apenas gradientes ou parâmetros dos modelos locais, sem acesso direto aos dados dos participantes, o que torna muitas defesas do aprendizado centralizado ineficazes. Como não é possível confiar plenamente nos participantes, o servidor precisa detectar atualizações anômalas ou reduzir a influência de valores atípicos durante a agregação, de modo que o modelo ainda consiga convergir mesmo quando uma fração dos clientes se comporta de forma maliciosa.

Uma primeira classe de defesas consiste em identificar e filtrar participantes maliciosos antes da agregação, como nos métodos AUROR [Shen et al., 2016], FoolsGold [Fung et al., 2018], FLTrust [Cao et al., 2020], Zeno++ [Xie et al., 2020] e COSDefense [Yaldiz et al., 2023]. O AUROR utiliza *k-means* para agrupar participantes com distribuições estatísticas similares nas características mais relevantes do modelo e descarta as atualizações fora desse agrupamento. O FoolsGold descarta atualizações de participantes que apresentam gradientes excessivamente semelhantes ao longo do tempo. O FLTrust explora um conjunto de dados confiável para calcular um "gradiente de referência" e remove participantes cujas atualizações se desviam demais desse gradiente segundo a similaridade de cosseno. O Zeno++ avalia as contribuições com base na variação da função de custo em um conjunto de validação, descartando atualizações que não melhoram ou degradam esse valor. O COSDefense analisa a similaridade de cosseno entre gradientes para identificar participantes alinhados em uma direção comum e descartar gradientes inconsistentes com essa direção.

Uma segunda classe de defesas modifica diretamente a regra de agregação para torná-la menos sensível a valores atípicos ou contribuições maliciosas, como em Krum [Blanchard et al., 2017], *Trimmed Mean* [Yin et al., 2018], Bulyan [Guerraoui et al., 2018] e *Robust Federated Aggregation* (RFA) [Pillutla et al., 2022]. O Krum calcula, para cada participante, a soma das distâncias euclidianas até seus vizinhos mais próximos e seleciona a atualização que é mais consistente com a maioria. O *Trimmed Mean* remove, coordenada a coordenada, os valores máximo e mínimo e calcula a média dos valores remanescentes. O Bulyan combina as ideias de Krum e *Trimmed Mean*: primeiro escolhe um subconjunto de atualizações mais confiáveis usando a regra de Krum e, em seguida, aplica o *Trimmed Mean* nesse subconjunto. O RFA utiliza a mediana geométrica em vez da média aritmética, buscando o ponto que minimiza a soma das distâncias a todas as atualizações e reduzindo, assim, o impacto de contribuições maliciosas.

Apesar de mitigarem ataques não-direcionados até certo ponto, essas técnicas de defesa não são suficientes contra ataques direcionados, como ataques *backdoor*, o que motiva o desenvolvimento de estratégias complementares, além da agregação robusta.

Diferentemente dos ataques não-direcionados, que buscam desestabilizar a convergência do modelo global, **ataques direcionados** (ataques de envenenamento *backdoor*) têm como objetivo manipular seletivamente o comportamento do modelo. Nesses cenários, as atualizações maliciosas são projetadas para se misturarem às atualizações benignas, de forma a permanecerem compatíveis com as estatísticas globais e, assim, burlar métodos agregação robusta e simples mecanismos de filtragem de anomalias. Por conta dessa natureza mais sutil, defesas contra ataques direcionados são tipicamente organizadas em duas famílias principais: (i) métodos de detecção, voltados a identificar a presença de um *backdoor*, e (ii) métodos de eliminação, voltados a remover ou mitigar o efeito do envenenamento já incorporado ao modelo [Li et al., 2022b].

Métodos de detecção exploram estatísticas das funções de ativação e propriedades estruturais do modelo para determinar se este foi envenenado por um ataque *backdoor*. A ideia central é que dados envenenados, ao serem incorporados ao treinamento, deixam traços característicos na representação latente aprendida, que podem ser diferenciados dos padrões produzidos por dados limpos. Tran *et al.* introduzem a noção de "assinatura espectral", segundo a qual amostras envenenadas produzem uma anomalia espectral na matriz de características internas do modelo [Tran et al., 2018]. Após o treinamento, constrói-se a matriz de covariância das representações e aplica-se *Single Value Decomposition* (SVD). As amostras envenenadas tendem a se projetar ao longo da direção de maior variância, permitindo identificá-las a partir dos componentes principais dominantes. Chen *et al.* propõem o *Activation Clustering* que parte da hipótese de que amostras legítimas e amostras envenenadas não geram o mesmo padrão de ativações nas camadas escondidas [Chen et al., 2018]. Nesse caso, coletam-se as saídas das funções de ativação para uma dada classe, aplica-se um algoritmo de agrupamento e verifica-se se as amostras dessa classe se organizam em dois grupos bem separados, dos quais um é associado às instâncias envenenadas.

Métodos de eliminação assumem que o ataque já foi bem-sucedido e visam remover o *backdoor* do modelo global treinado, reduzindo ou anulando o comportamento malicioso sem destruir completamente a capacidade do modelo na tarefa principal. Liu *et*

al. propõem o *Fine-Pruning*, que explora o fato de que o *backdoor* tende a ser codificado em neurônios pouco ativados por amostras legítimas, mas fortemente ativados na presença do gatilho [Liu et al., 2018]. A técnica mede a ativação média de cada neurônio sob dados limpos, poda aqueles com baixa ativação e realiza um ajuste fino (*fine-tuning*) com os neurônios restantes, buscando preservar o desempenho legítimo enquanto remove a lógica associada ao *backdoor*. Li *et al.* introduzem o *Neural Attention Distillation* (NAD), que utiliza um modelo instrutor, treinado em dados confiáveis, para guiar a recuperação de um modelo aprendiz envenenado [Li et al., 2021b]. O método alinha os mapas de atenção das camadas escondidas do aprendiz com os do instrutor, incentivando o modelo comprometido a concentrar-se novamente em padrões semânticos legítimos e a deixar de depender dos padrões de acionamento do *backdoor*.

Outra linha é o *Mode Connectivity Repair* (MCR), proposto por Zhao *et al.*, que se apoia em propriedades geométricas da paisagem de perda de redes neurais [Zhao et al., 2020b]. A premissa é que um modelo treinado apenas com dados legítimos e um modelo envenenado correspondem a dois pontos no espaço de parâmetros que podem ser conectados por um caminho de baixa perda. Ao construir explicitamente esse caminho e otimizar o valor esperado da perda ao longo da curva, é possível deslocar o modelo envenenado em direção a uma região "saudável" da paisagem de perda, reparando o comportamento adversarial sem reiniciar o treinamento do zero. Finalmente, o *Anti-Backdoor Learning* (ABL), também proposto por Li *et al.*, explora o fato de que padrões associados ao *backdoor* são aprendidos mais rapidamente e normalmente concentrados em uma classe específica [Li et al., 2021a]. O ABL aplica gradiente ascendente nas amostra, monitorando a evolução da perda para distinguir exemplos que apresentam rápido decaimento (típico de instâncias envenenadas) daqueles com dinâmica mais gradual (típico de exemplos legítimos). Uma vez separadas, aplica-se gradiente ascendente sobre o conjunto de amostras classificadas como envenenadas, forçando o modelo a desaprender o comportamento de *backdoor* associado a essas instâncias.

Em conjunto, essas estratégias mostram que a defesa contra ataques direcionados exige ir além da mera robustez de agregação: é necessário analisar representações internas, dinâmica de treinamento e geometria da superfície de perda para identificar e remover, de forma seletiva, os comportamentos maliciosos inseridos no modelo. A Tabela 1.1 sintetiza os principais tipos de ataques em Aprendizado Federado e as respectivas estratégias de mitigação discutidas nas seções anteriores.

1.5. Atividade Prática

Essa seção descreve as atividades práticas, a implementação de ataques relevantes em Aprendizado Federado utilizando o arcabouço Flower, na versão 1.25.0, em um ambiente experimental controlado. O objetivo é permitir que os participantes compreendam, na prática, como diferentes classes de ataques podem ser implementadas, executadas e avaliadas em um cenário federado.

A atividade foi estruturada em torno dos dois tipos de ataques ao FL, os ataques ao desempenho do modelo e os ataques à privacidade. No primeiro grupo, são implementados três ataques bizantinos, *A Little Is Enough* (ALIE), *Sign Flipping* e injeção de ruído gaussiano, com o propósito de mostrar como clientes maliciosos podem manipular

Tabela 1.1. Tipos de ataques em Aprendizado Federado e estratégias de mitigação associadas.

Tipo de ataque	Subtipo	Mecanismo de ação	Defesa principal	Mecanismo de defesa
Envenenamento do Modelo Ataques Bizantinos	Envenenamento de dados	Cliente altera rótulos, atributos ou insere exemplos sintéticos para enviesar gradientes locais e degradar o modelo global.	Agregação robusta (Krum, *Trimmed Mean*, Bulyan, RFA) [Blanchard et al., 2017, Yin et al., 2018, Guerraoui et al., 2018, Pillutla et al., 2022]	Ajusta a regra de agregação para reduzir o peso de valores atípicos e atualizações maliciosas.
	Envenenamento de modelo	Cliente manipula diretamente direção e norma da atualização enviada ao servidor para desviar a trajetória de otimização.	Filtragem de clientes (AUROR, FoolsGold, FLTrust, Zeno++, COSDefense) [Shen et al., 2016, Fung et al., 2018, Cao et al., 2020, Xie et al., 2020, Yaldiz et al., 2023]	Detecta e descarta atualizações suspeitas com base em similaridade estatística, gradientes ou impacto em validação.
	Bizantino descentralizado	Clientes desonestos difundem modelos locais corrompidos na vizinhança em topologias sem servidor confiável.	Agregação robusta descentralizada [Fang et al., 2024]	Usa regras robustas de consenso entre vizinhos para limitar a propagação de atualizações anômalas.
Ataques à privacidade	Inferência de participação	Adversário estima se uma amostra específica participou do treinamento a partir de saídas, gradientes ou diferenças de parâmetros.	Privacidade diferencial (CDP, LDP, DDP) [McMahan et al., 2018, Wang et al., 2019, Agarwal et al., 2018, Lyu, 2020]	Injeta ruído calibrado em parâmetros ou atualizações para limitar a influência de cada exemplo no modelo.
	Inferência de atributos	Adversário infere propriedades latentes ou sensíveis dos dados com base em atualizações e representações internas.	LDP/DDP com ruído local e distribuído [Wang et al., 2019, Agarwal et al., 2018]	Perturba gradientes locais para mascarar correlações entre atributos privados e parâmetros.
	Inferência de origem	Adversário vincula registros ou rótulos a um cliente específico usando respostas de modelos locais.	Aprendizado Federado vertical protegido e SMC [Fu et al., 2022, Agarwal et al., 2018]	Emprega computação segura e separação de atributos/rótulos para reduzir o vazamento de autoria dos dados.
	Reconstrução de conteúdo (DLG, iDLG, GAN)	Adversário reconstrói exemplos privados (imagens, textos) a partir de gradientes ou da evolução do modelo.	Privacidade diferencial e criptografia homomórfica [Zhu et al., 2019, Zhao et al., 2020a, Lim et al., 2020, de Assis et al., 2023, de Assis et al., 2024]	Adiciona ruído e realiza agregação sobre dados cifrados para limitar a informação útil à reconstrução.
Ataques direcionados *backdoor*	Backdoor por dados com gatilho	Cliente insere exemplos com padrão de gatilho e rótulo alvo para ativar saídas maliciosas sob condições específicas.	Detecção por representação (assinatura espectral, agrupamento de ativação) [Tran et al., 2018, Chen et al., 2018]	Analisa representações internas para identificar subgrupos de exemplos com padrões latentes discrepantes associados ao *backdoor*.
	Substituição de modelo (*model replacement*)	Cliente reescala sua atualização para aproximar o modelo global de um modelo local envenenado, mantendo boa acurácia padrão.	Agregação robusta e limitação de norma [Wang et al., 2020, Pillutla et al., 2022]	Impõe limites de norma e regras robustas para evitar que uma única atualização domine o modelo global.
	Backdoor por casos raros	Cliente usa exemplos raros, porém legítimos, associados a um rótulo alvo para esconder o gatilho em regiões pouco amostradas.	Fine-Pruning, NAD, MCR, ABL [Liu et al., 2018, Li et al., 2021b, Zhao et al., 2020b, Li et al., 2021a]	Remove ou enfraquece o *backdoor* podando neurônios, realinhando atenção, explorando conectividade de modos e desaprendendo padrões maliciosos.
Ataques ao protocolo	Servidor curioso	Servidor tenta inferir dados sensíveis a partir das atualizações recebidas ou quebra suposições de honestidade.	Criptografia homomórfica e SMC [Lim et al., 2020, de Assis et al., 2023, de Assis et al., 2024, Agarwal et al., 2018]	Permite agregação sobre dados cifrados ou via computação segura, ocultando atualizações em texto claro.
	Exploração de heterogeneidade / *distributional shift*	Adversário explora a variabilidade natural dos dados e mudanças de distribuição para camuflar atualizações maliciosas.	Agregação robusta com validação de referência [Zhang et al., 2025, Li et al., 2023]	Combina regras robustas com validação em dados de referência para separar heterogeneidade benigna de desvios adversariais.

atualizações locais para degradar a convergência e o desempenho do modelo global. No segundo grupo, são implementados dois ataques voltados à privacidade, *Deep Leakage from Gradients* (DLG) e reconstrução baseada em GAN, evidenciando que o compartilhamento de gradientes ou atualizações pode expor informações sensíveis sobre clientes.

1.5.1. Configuração do Cenário do Aprendizado Federado no Flower

O Flower é um arcabouço de Aprendizado Federado que organiza a execução em dois componentes centrais: o `ServerApp`, responsável pela estratégia e pela orquestração do treinamento, e o `ClientApp`, responsável pelo treinamento e pela avaliação local em cada cliente. A comunicação entre servidor e clientes é realizada por meio de mensagens que transportam um `RecordDict`, estrutura de dados que pode conter diferentes tipos de registros, como `ArrayRecord`, `MetricRecord` e `ConfigRecord`. O `ArrayRecord` é utilizado para armazenar estruturas numéricas serializadas, tipicamente associadas aos parâmetros do modelo ou a outros arranjos numéricos relevantes para o treinamento. Na prática, ele é usado para encapsular os parâmetros do modelo global enviados pelo servidor aos clientes e, posteriormente, os parâmetros ou atualizações locais devolvidos pelos clientes ao servidor.

Em aplicações com PyTorch, o `ArrayRecord` pode ser construído diretamente a partir de um `state_dict` e convertido novamente para esse formato no momento da execução local. Por essa razão, o `ArrayRecord` constitui o principal mecanismo para transportar o estado do modelo ao longo das rodadas de comunicação. O `MetricRecord`, por sua vez, é utilizado para armazenar métricas numéricas produzidas durante o treinamento ou a avaliação. Essas métricas podem incluir, por exemplo, perda do treinamento, acurácia, número de exemplos processados e outros valores de interesse para monitoramento ou agregação. Já o `ConfigRecord` é utilizado para transportar parâmetros de configuração entre servidor e clientes.

Para iniciar um novo ambiente no Flower, utiliza-se o comando `flwr new`, que cria automaticamente um esqueleto inicial do projeto. Uma instalação típica do ambiente de simulação pode ser iniciada com os comandos do Código 1.1. O projeto gerado já organiza a aplicação em arquivos como `client_app.py`, `server_app.py` e `task.py`. O arquivo `client_app.py` contém a lógica executada pelos clientes, isto é, o treinamento e a avaliação local. O arquivo `server_app.py` contém a lógica do servidor, incluindo a definição do protocolo de agregação, chamado de *Strategy* pelo Flower, e a coordenação das rodadas de treinamento. Já o arquivo `task.py` concentra elementos compartilhados pela aplicação, como definição do modelo, carregamento do conjunto de dados, particionamento dos dados e funções auxiliares de treino e teste.

Os arquivos gerados constituem um exemplo simples de Aprendizado Federado que pode ser facilmente modificado para cenários mais complexos alterando poucas linhas de código. Para modificar o conjunto de dados, basta alterar a parte de carregamento e particionamento dos dados no arquivo `task.py`, onde são definidos o conjunto de dados utilizado, o pré-processamento aplicado e o particionador responsável por distribuir as amostras entre os clientes. Para mudar a distribuição dos dados dos clientes, basta substituir ou reconfigurar o particionador utilizado no `task.py`. Em particular, pode-se usar o `IidPartitioner` para simular um cenário IID ou o `DirichletPartitioner`

para simular um cenário *non-IID*, ajustando parâmetros como o número de partições e o valor de `alpha`, que controla o grau de heterogeneidade entre os clientes. Para mudar as configurações da simulação, como número de clientes, número de rodadas, épocas locais, taxa de aprendizado, tamanho do lote e outros hiperparâmetros globais, basta modificar o arquivo de configuração `pyproject.toml`. Nesse arquivo, as opções da simulação são definidas na seção de configuração da aplicação Flower e ficam disponíveis durante a execução por meio de `context.run_config` e `context.node_config`.

Após a criação do projeto, a instalação local pode ser concluída com `pip install -e .` e a execução da simulação pode ser realizada por meio de `flwr run .`.

```
pip install -U "flwr[simulation]"
flwr new fl-cifar100-attacks --framework pytorch --username <
    seu_usuario>
cd fl-cifar100-attacks
pip install -e .
flwr run .
```

Código Fonte 1.1. Inicialização de um novo projeto Flower

Com essa configuração, estabelece-se uma base experimental apropriada para a implementação dos ataques. O Flower fornece a infraestrutura de comunicação e orquestração e os particionamentos *IID* e *non-IID* permitem investigar o comportamento dos ataques sob diferentes graus de heterogeneidade estatística. A partir desse cenário, torna-se possível introduzir, de forma controlada, tanto ataques bizantinos voltados à degradação do desempenho do modelo global quanto ataques de privacidade voltados à extração ou reconstrução de informações sensíveis.

1.5.2. Modelo do Atacante

Na atividade prática, os ataques são organizados segundo dois modelos de adversário distintos. Para os ataques ao desempenho do modelo global, adota-se um modelo em que o atacante atua como um cliente participante do treinamento de FL. Para os ataques à privacidade, adotam-se dois modelos de adversário complementares. No caso do DLG, o adversário corresponde ao próprio servidor agregador, assumido como *honesto-mas-curioso*. Já no caso da reconstrução com GAN, o adversário é modelado como um cliente participante curioso, que segue o protocolo federado, mas explora as informações disponíveis durante o treinamento para inferir conteúdo privado de outro participante. A adoção desses dois modelos distintos tem como objetivo enriquecer a demonstração prática, mostrando que os riscos à privacidade em Aprendizado Federado podem emergir tanto do lado do servidor quanto do lado dos próprios clientes participantes.

No caso dos ataques bizantinos, o atacante é modelado como um cliente participante do sistema de Aprendizado Federado, isto é, como um cliente que pode ou não ser selecionado para participar das rodadas de treinamento, com acesso normal ao modelo global enviado pelo servidor e com capacidade de retornar uma atualização local manipulada se selecionado. Esse atacante segue o fluxo esperado do protocolo de agregação no sentido de receber o modelo, executar localmente seu procedimento e responder ao servidor no momento previsto. Entretanto, em vez de enviar uma contribuição compatível com o treinamento benigno, esse cliente envia uma atualização adulterada com o objetivo

de degradar a convergência, reduzir o desempenho final ou desestabilizar o treinamento global. Esse modelo é consistente com a caracterização de ataques bizantinos discutida no texto, em que clientes maliciosos se desviam do comportamento esperado e enviam atualizações adversariais ao processo de agregação. No contexto deste trabalho, os ataques ALIE, *Sign Flipping* e injeção de ruído gaussiano seguem exatamente essa lógica. Em todos esses casos, o adversário não precisa controlar o servidor nem comprometer toda a federação. Basta que um ou mais clientes participantes estejam sob controle malicioso para que as atualizações enviadas ao agregador passem a induzir desvios na trajetória de otimização do modelo global.

Já nos ataques à privacidade, o modelo adversarial adotado é diferente. Nesse caso, assume-se que o atacante pode ser o servidor agregador e que esse servidor opera sob o modelo *honesto-mas-curioso*, ou um participante curioso. Um servidor *honesto-mas-curioso* é uma entidade que executa corretamente o protocolo de FL, isto é, envia os parâmetros esperados aos clientes, recebe as respostas e realiza a agregação conforme definido, sem alterar explicitamente o fluxo da execução. No entanto, embora siga o protocolo de forma correta, esse servidor tenta extrair o máximo possível de informação sensível a partir dos dados que observa durante o treinamento, como gradientes, parâmetros locais, diferenças entre parâmetros ou outras atualizações recebidas dos clientes. Em outras palavras, trata-se de um adversário que é honesto do ponto de vista operacional, mas curioso do ponto de vista informacional. Esses modelos são amplamente utilizados na análise de privacidade em Aprendizado Federado porque refletem um cenário em que o principal risco não decorre da quebra explícita do protocolo, mas da possibilidade de inferência a partir das informações legitimamente acessíveis ao agregador [Lim et al., 2020, Yang et al., 2019]. Esses modelos são adequados para os ataques DLG e reconstrução com GAN considerados nesta atividade prática.

1.5.3. Implementação dos Ataques ao Desempenho do Modelo

Os ataques bizantinos são implementados no arquivo `client_app.py`. O código fonte encontra-se disponível no GitLab[7] A ideia central da implementação dessa simulação é que cada cliente recebe o modelo global enviado pelo servidor, realiza o treinamento local normalmente e, em seguida, caso seja identificado como malicioso, aplica uma transformação adversarial sobre os parâmetros do modelo antes de devolvê-los ao servidor. Com isso, o ataque não altera a estrutura do protocolo de comunicação do Flower, mas interfere diretamente no conteúdo da atualização enviada ao agregador.

Para esse fim, foram criadas três funções específicas em `client_app.py`: `apply_gaussian_noise_attack()`, `apply_alie_attack()` e `apply_sign_flip_attack()`. A função `apply_gaussian_noise_attack()` implementa o ataque por ruído gaussiano. Nesse ataque, o cliente malicioso adiciona perturbações aleatórias aos parâmetros do modelo treinado localmente. O objetivo é corromper a atualização local sem seguir uma direção de otimização consistente, inserindo ruído suficiente para degradar a qualidade da agregação global.

[7]Disponível em `https://gitlab.com/LCC/fl-performance-attack`. Acessado em 21/04/2026

```python
def apply_gaussian_noise_attack(model: torch.nn.Module, noise_std: float = 1.0) -> None:
    with torch.no_grad():
        for param in model.parameters():
            noise = torch.randn_like(param) * noise_std
            param.add_(noise)
```

Código Fonte 1.2. Implementação do ataque por ruído gaussiano

A função recebe como entrada o modelo local e um parâmetro `noise_std`, que controla o desvio padrão do ruído gaussiano. O bloco `with torch.no_grad()` é utilizado para garantir que essa modificação não seja registrada pelo mecanismo de diferenciação automática do PyTorch, já que o objetivo não é realizar retropropagação, mas apenas alterar diretamente os valores numéricos dos parâmetros. Em seguida, a função percorre todos os parâmetros do modelo. Para cada parâmetro, é gerado um tensor de ruído com a mesma dimensão do tensor original por meio de `torch.randn_like(param)`, sendo esse ruído escalado pelo fator `noise_std`. Por fim, o ruído é somado diretamente ao parâmetro com `param.add_(noise)`.

Do ponto de vista do ataque, essa implementação produz uma atualização local ruidosa, cuja direção não corresponde à direção benigna da otimização. Assim, quando essa atualização é agregada juntamente com atualizações benignas, ela pode deslocar o modelo global para distantes do mínimo global da função de perda, prejudicando a convergência e a acurácia final.

A função `apply_alie_attack()` implementa o ataque *A Little Is Enough* (ALIE). Diferentemente do ruído gaussiano, o ALIE procura construir uma atualização maliciosa com aparência estatisticamente plausível. A ideia é produzir uma atualização desviada, mas sem gerar uma anomalia extremamente evidente, dificultando a distinção entre comportamento benigno e comportamento malicioso.

```python
def apply_alie_attack(
    model: torch.nn.Module,
    global_state_dict: dict,
    num_clients: int,
    num_malicious: int,
    z_max: float | None = None,) -> None:

    from scipy.stats import norm

    n, m = num_clients, num_malicious

    if z_max is None:
        s = max(1, (n // 2 + 1) - m)
        p = (n - m - s) / max(n - m, 1)
        p = min(max(p, 1e-6), 1 - 1e-6)
        z_max = float(norm.ppf(p))

    with torch.no_grad():
        for (name, param), global_param in zip(
            model.named_parameters(), global_state_dict.values()
        ):
            global_param = global_param.to(param.device)
            mu = global_param.mean()
```

```
24              sigma = global_param.std()
25              low = mu - z_max * sigma
26              high = mu + z_max * sigma
27              param.data = torch.empty_like(param.data).uniform_(low.item
    (), high.item())
```

Código Fonte 1.3. Implementação do ataque ALIE

A função recebe o modelo local já treinado e o *state_dict* global, o número total de clientes, o número de clientes maliciosos e, opcionalmente, o parâmetro `z_max`. No Código 1.3 linhas 13 — 17, há uma estrutura para cálculo automático do `z_max` a partir de uma aproximação baseada na quantidade total de clientes lícitos e clientes maliciosos.

Após essa etapa, o código percorre simultaneamente os parâmetros do modelo local e os correspondentes do modelo global recebido do servidor. A seguir, a função calcula duas estatísticas escalares do parâmetro global, a média `mu = global_param.mean()` e o desvio-padrão `sigma = global_param.std()`. Como `global_param` é um tensor, essas operações retornam valores agregados sobre todos os elementos daquele tensor. Assim, a média representa o valor médio dos parâmetros da camada global, enquanto o desvio-padrão representa a dispersão global.

Com essas estatísticas, a função define um intervalo de amostragem $w_j \sim \mathcal{U}(\mu - z_{\max}\sigma,\ \mu + z_{\max}\sigma)$. Esse intervalo representa uma faixa centrada na média dos parâmetros globais e com largura proporcional ao desvio padrão escalado por `z_max`. Em seguida, o parâmetro local é completamente sobrescrito por novos valores gerados aleatoriamente de forma uniforme nesse intervalo e atribuído a cada parâmetro do modelo, por meio da instrução da linha 27.

A função `apply_sign_flip_attack()` implementa o ataque *Sign Flip*. Nesse caso, o objetivo é inverter o sinal de parâmetros do modelo local, direcionando a atualização na direção oposta àquela produzida pelo treinamento benigno. Trata-se de uma implementação direta de um ataque de envenenamento do modelo, no qual a contribuição enviada ao servidor passa a contrariar o processo de descida da função de perda.

```
1  def apply_sign_flip_attack(
2      model: torch.nn.Module,
3      flip_factor: float = -1.0,
4      top_fraction: float = 0.2,
5  ) -> None:
6      with torch.no_grad():
7          for param in model.parameters():
8              if top_fraction >= 1.0:
9                  param.mul_(flip_factor)
10             else:
11                 flat = param.data.view(-1)
12                 k = max(1, int(flat.numel() * top_fraction))
13                 _, top_indices = torch.topk(flat.abs(), k)
14                 flat[top_indices] *= flip_factor
```

Código Fonte 1.4. Implementação do ataque Sign Flip

A função recebe o modelo local, o fator de inversão `flip_factor` e a fração `top_fraction`. O valor padrão de `flip_factor` é -1.0, o que significa inverter o sinal dos parâmetros selecionados. Já `top_fraction` controla a fração de parâmetros

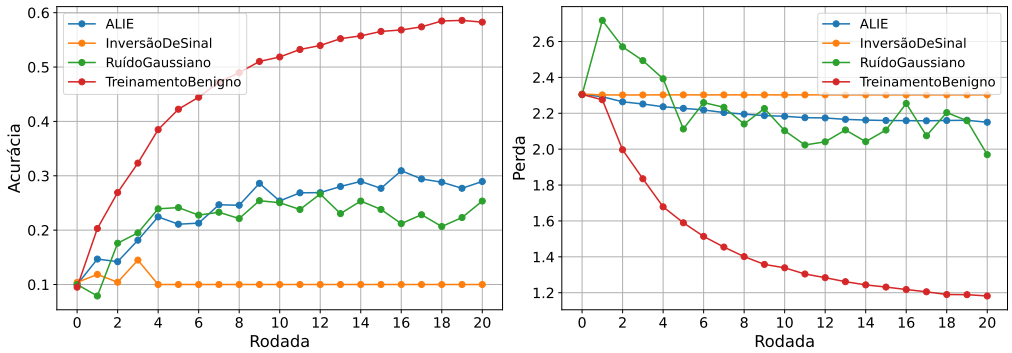


Figura 1.5. Evolução da acurácia e da perda ao longo das rodadas de agregação com protocolo FedAvg, comparando o treinamento benigno com os ataques. O gráfico da esquerda mostra a acurácia global por rodada, enquanto o gráfico da direita mostra a perda global correspondente.

que sofrerá inversão. Se `top_fraction >= 1.0`, todos os parâmetros da camada têm seus sinais invertidos. Caso contrário, a função seleciona apenas os parâmetros de maior magnitude absoluta dentro de cada tensor de parâmetros.

Essa seleção parcial é realizada achatando o tensor com `view(-1)`, calculando a quantidade k de elementos a serem alterados e aplicando `torch.topk(flat.abs(), k)` para obter os índices dos maiores valores absolutos. Em seguida, apenas esses elementos têm seus sinais invertidos. Essa implementação permite um ataque menos uniforme e potencialmente mais furtivo, concentrando a manipulação nos parâmetros de maior magnitude, que tendem a ter maior influência sobre a atualização final do modelo.

A aplicação efetiva dos ataques ocorre na função `train()`, que representa a rotina de treinamento local executada pelos clientes. Essa função não apenas realiza o treino benigno, mas também determina quais clientes serão maliciosos e o ataque.

Após o treinamento local, a função passa a decidir se aquele cliente será benigno ou malicioso. Para isso, são utilizados dois elementos do arquivo de configuração da aplicação. O primeiro é `attack-type`, que define qual ataque será aplicado, e o segundo é o parâmetro `malicious-fraction`, recuperados por meio de `context.run_config`. O parâmetro `malicious-fraction` indica a fração de clientes da federação que será tratada como maliciosa.

Essa lógica significa que os primeiros clientes, de acordo com o identificador de partição, são marcados como maliciosos. Por exemplo, se houver 10 clientes e `malicious-fraction = 0.2`, então `num_malicious = 2`, e os clientes com `partition_id` 0 e 1 serão considerados atacantes. Trata-se de uma escolha de implementação simples e determinística, suficiente para experimentos controlados.

Para tornar a simulação mais flexível, foram introduzidos o parâmetro `malicious-fraction` e `attack-type` no arquivo de configuração da aplicação (`pyproject.toml`), acessado em tempo de execução por meio de `context.run_config`. Esse parâmetro permite controlar, sem necessidade de alterar o código-fonte da função `train()`, qual fração dos clientes participantes será configurada como maliciosa em cada experimento.

A Figura 1.5 apresenta a evolução da acurácia e da perda do modelo global ao longo das rodadas de treinamento federado utilizando o protocolo de agregação FedAvg.

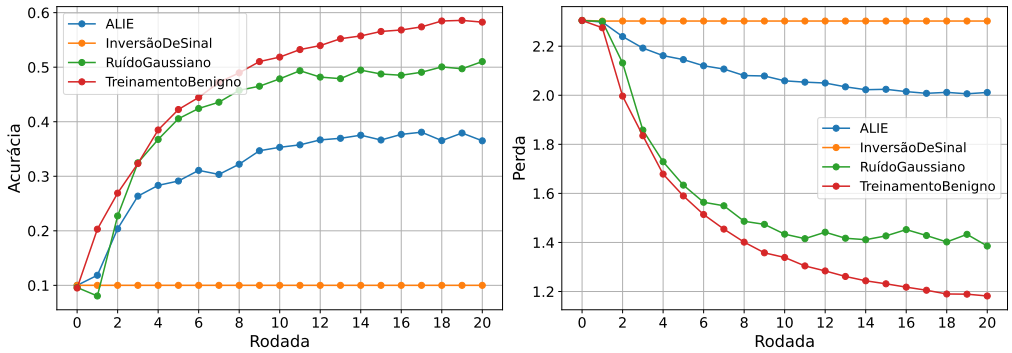


Figura 1.6. Evolução da acurácia e da perda ao longo das rodadas de treinamento sob o protocolo de agregação TrimmedMean, comparando o treinamento benigno com os ataques. Observa-se que a inversão de sinal compromete severamente a convergência, mantendo a acurácia próxima ao nível inicial e a perda praticamente constante, enquanto ALIE também degrada o treinamento de forma significativa. Em contraste, o ruído gaussiano produz impacto intermediário, e o treinamento benigno apresenta o melhor comportamento global.

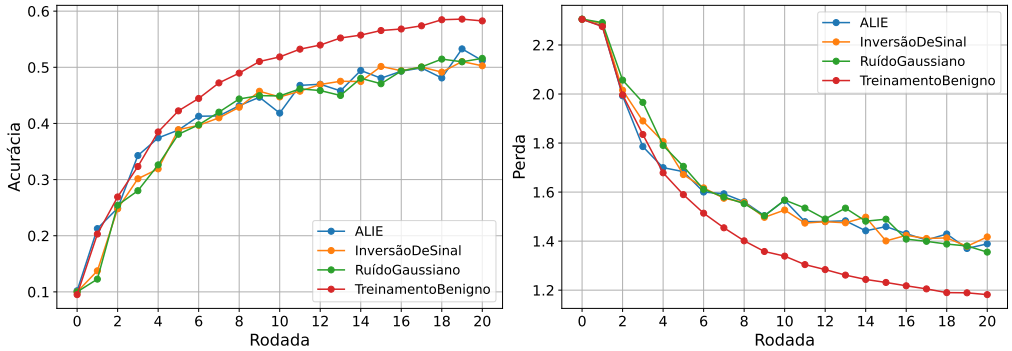


Figura 1.7. Evolução da acurácia e da perda ao longo das rodadas de treinamento sob o protocolo de agregação MultiKrum, comparando o treinamento benigno com os ataques. Diferentemente do observado com TrimmedMean, o MultiKrum reduz substancialmente o impacto dos ataques, permitindo que os cenários maliciosos mantenham trajetórias de convergência mais próximas entre si e mais próximas do treinamento benigno, ainda que com desempenho final inferior.

Observa-se que, no cenário benigno, o modelo apresenta crescimento progressivo da acurácia e redução consistente da perda, indicando convergência estável do treinamento. Em contraste, os ataques comprometem esse comportamento em diferentes níveis. O ataque por inversão de sinal é o mais severo, mantendo a acurácia próxima a 0,1 e a perda praticamente estável em valores elevados, o que indica forte comprometimento da aprendizagem global. O ataque por ruído gaussiano também degrada significativamente o desempenho, produzindo oscilações mais acentuadas e limitando o avanço da acurácia. Já o ataque ALIE, embora menos destrutivo que os demais, ainda reduz o desempenho em relação ao treinamento benigno, com acurácia inferior e perda superior ao longo das rodadas.

Na Figura 1.6, observa-se que os ataques afetam o treinamento de maneira bastante distinta. O ataque por inversão de sinal é o mais severo, pois mantém a acurácia praticamente estagnada em torno de 0,1 ao longo de todas as rodadas, ao mesmo tempo em que a perda permanece elevada e quase constante, indicando falha de convergência. O ataque ALIE também compromete significativamente o aprendizado, reduzindo a taxa de crescimento da acurácia e mantendo a perda em níveis substancialmente superiores aos do treinamento benigno. Já o ruído gaussiano produz um efeito menos destrutivo, embora

degrade o desempenho em relação ao caso benigno, ainda permite evolução do treinamento, com acurácia final superior à obtida com ALIE e perda final inferior à observada nos cenários mais agressivos.

Na Figura 1.7, o efeito dos ataques é significativamente atenuado em comparação ao caso com TrimmedMean. Todos os cenários maliciosos apresentam crescimento progressivo de acurácia e redução consistente da perda, indicando que o treinamento continua convergente mesmo sob comportamento adversarial. O treinamento benigno ainda apresenta o melhor desempenho final, com maior acurácia e menor perda, mas a distância em relação aos cenários com participantes adversariais torna-se bem menor comparada aos demais cenários.

1.5.4. Implementação dos Ataques à Privacidade

Esta seção apresenta a implementação prática dos ataques à privacidade considerados, com o objetivo de demonstrar, em ambiente controlado, como informações sensíveis podem ser inferidas a partir dos elementos compartilhados durante o treinamento federado. No ataque DLG, considera-se um servidor agregador *honesto-mas-curioso*, enquanto, na reconstrução com GAN, considera-se um cliente participante curioso.

O ataque DLG tem como objetivo reconstruir uma amostra privada de treinamento a partir dos gradientes compartilhados durante o Aprendizado Federado. O código fonte encontra-se disponível no GitLab[8]. Para executar o comportamento do servidor honesto-mas-curioso no Flower, foi criada uma nova *Strategy*, denominada `MaliciousFedAvg`, que preserva a lógica do *FedAvg*, mas acrescenta os passos do ataque antes da agregação final. A principal modificação foi feita no método `aggregate_train()`.Esse método continua responsável por agregar as respostas dos clientes, mas passa a incluir também a seleção de uma vítima, a recuperação de seus gradientes e a chamada da rotina de reconstrução DLG.

Ainda na função `aggregate_train()`, o servidor agregador verifica respostas recebidas e compara o estado do modelo global antes e depois do treinamento local. O servidor escolhe aleatoriamente um cliente como vítima, e armazena a resposta. Com isso, a implementação simula um servidor que observa normalmente todas as respostas, mas decide explorar apenas uma delas em cada rodada.

O servidor, calcula

$$g = \theta_{\text{server}} - \theta_{\text{victim}},$$

isto é, a diferença entre os parâmetros do modelo global enviados no início da rodada e os parâmetros devolvidos pelo cliente após o treinamento local. Essa etapa materializa a hipótese central do ataque: se o servidor conhece o modelo enviado e o modelo recebido, então o servidor também consegue reconstruir a atualização produzida pelo cliente, e essa atualização pode ser explorada para inferir informação privada.

O servidor chama a função `dlg_attack_from_gradients()`, função que implementa o ataque DLG, fornecendo a rede, os gradientes recuperados, a forma esperada da entrada, o número de classes, o número de iterações do *Limited-memory Broyden-Fletcher-Goldfarb-Shannon* (L-BFGS), o diretório de saída e o identificador

[8]Disponível em `https://gitlab.com/LCC/fl-dlg-attack`. Acessado em 21/04/2026.

da vítima. O L-BFGS é um método de otimização empregado para ajustar iterativamente a entrada fictícia do ataque DLG, de modo que os gradientes produzidos por essa entrada se aproximem dos gradientes reais observados [Liu e Nocedal, 1989]. Finalmente,a implementação retorna `super().aggregate_train(server_round, iter(replies_list))`, o que significa que, após executar o ataque, o servidor prossegue com a agregação padrão do *FedAvg*.

A rotina de reconstrução propriamente dita é implementada na função `dlg_attack_from_gradients()`, apresentada no Código 1.5. Essa função recebe como entrada o modelo já carregado com os parâmetros globais e a lista de gradientes recuperados da vítima, e tenta reconstruir uma entrada fictícia cuja passagem pela rede produza gradientes o mais próximos possível dos gradientes observados.

```
def dlg_attack_from_gradients(
    net, original_dy_dx,
    input_shape=(1, 3, 32, 32), num_classes=100,
    num_iterations=300, save_dir="dlg_results", client_id=None,
):
    tag = f"client_{client_id}" if client_id is not None else "victim"
    save_dir = os.path.join(save_dir, tag)
    os.makedirs(save_dir, exist_ok=True)

    device = next(net.parameters()).device
    criterion = cross_entropy_for_onehot

    print(f"\n{'='*60}")
    print(f"  Ataque DLG em {tag} - {num_iterations} iteracoes L-BFGS")
    print(f"{'='*60}")

    dummy_data = torch.randn(input_shape).to(device).requires_grad_(
    True)
    dummy_label = torch.randn((1, num_classes)).to(device).
    requires_grad_(True)
    initial_noise = tt(dummy_data[0].cpu().detach())

    optimizer = torch.optim.LBFGS([dummy_data, dummy_label])
    history, losses = [], []

    for iters in range(num_iterations):
        def closure():
            optimizer.zero_grad()
            pred = net(dummy_data)
            dummy_onehot_label = F.softmax(dummy_label, dim=-1)
            dummy_loss = criterion(pred, dummy_onehot_label)
            dummy_dy_dx = torch.autograd.grad(
                dummy_loss, net.parameters(), create_graph=True)
            grad_diff = 0
            for gx, gy in zip(dummy_dy_dx, original_dy_dx):
                grad_diff += ((gx - gy) ** 2).sum()
            grad_diff.backward()
            return grad_diff

        optimizer.step(closure)
        current_loss = closure()
```

```
40          losses.append(current_loss.item())
41          if iters % 10 == 0:
42              print(f"  {tag} | iter {iters:>3d} | loss {current_loss.
    item():.4f}")
43          history.append(tt(dummy_data[0].cpu()))
44
45      recovered_label = torch.argmax(dummy_label, dim=-1).item()
46      print(f"  {tag} | Label recuperada: {recovered_label}")
47
48      _save_results(history, losses, initial_noise, None, save_dir,
49                    f"Reconstrucao DLG - {tag}")
50      print(f"  Resultados salvos em: {os.path.abspath(save_dir)}/")
51      return dummy_data.detach(), dummy_label.detach()
```

Código Fonte 1.5. Função `dlg_attack_from_gradients()`

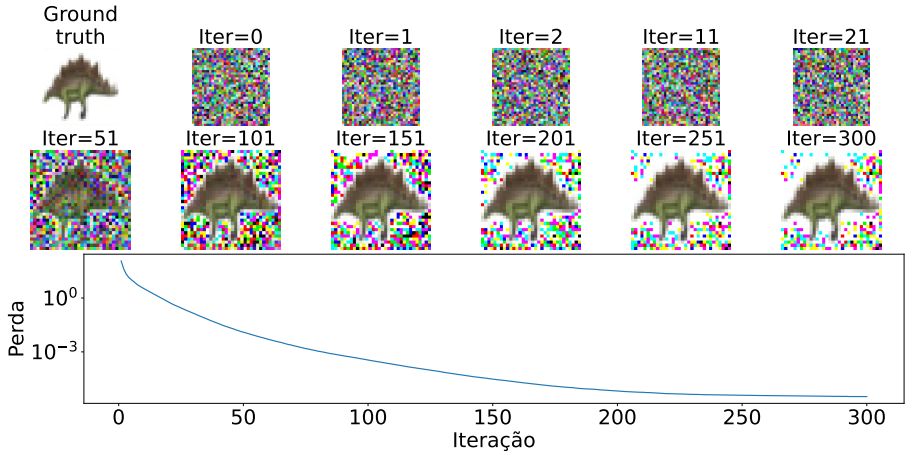
A função `dlg_attack_from_gradients()`, apresentada no Código 1.5, recebe como entrada o modelo e os gradientes recuperados e resolve o problema inverso de reconstrução. O primeiro ponto central da implementação está nas linhas 17 e 18, em que o ataque inicializa uma imagem fictícia e um rótulo fictício, ambos aleatórios, e os transforma em variáveis otimizáveis. O objetivo do DLG não é ajustar os parâmetros da rede, mas ajustar essa entrada fictícia até que ela produza gradientes compatíveis com os gradientes observados.

Na linha 21, a implementação define o L-BFGS como método de otimização. Esse otimizador é usado para atualizar simultaneamente a imagem e o rótulo fictícios ao longo das iterações do ataque. Na linha 27, a imagem fictícia atual é passada pela rede. Na linha 28, o rótulo fictício é convertido em uma distribuição válida. Na linha 29, calcula-se a perda associada a essa entrada fictícia. O ponto mais importante aparece nas linhas 30 a 32, em que o ataque calcula os gradientes fictícios da imagem atualmente reconstruída em relação aos parâmetros da rede. O argumento `create_graph=True` é indispensável, pois permite diferenciar posteriormente a discrepância entre esses gradientes fictícios e os gradientes reais em relação à própria imagem fictícia.

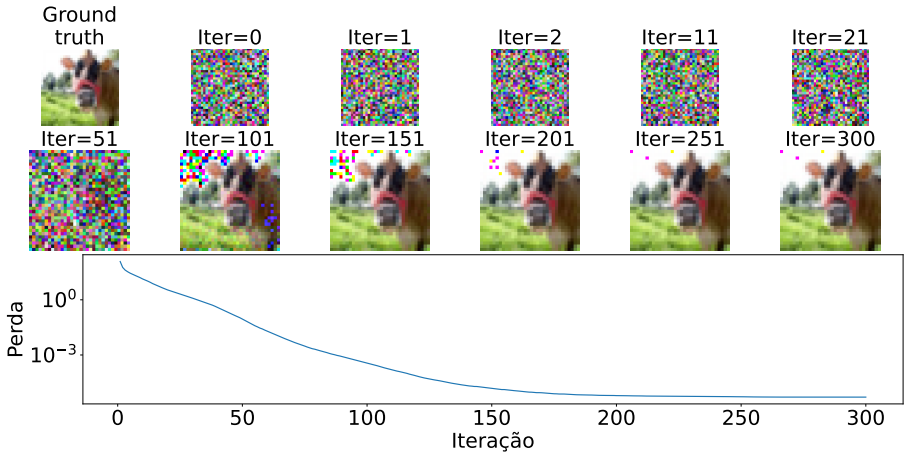
Em seguida, nas linhas 33 a 35, o código define a função objetivo do ataque, acumulando a soma das diferenças quadráticas entre os gradientes fictícios e os gradientes reais observados. Portanto, o DLG pode ser entendido como um problema de otimização em que se busca minimizar a discrepância entre dois conjuntos de gradientes. Na linha 36, a discrepância calculada é retropropagada até `dummy_data` e `dummy_label`. Esse é o mecanismo que ajusta a reconstrução, a cada iteração, a imagem fictícia é modificada na direção que reduz a diferença entre os gradientes que ela produz e os reais da vítima.

Fora da `closure()`, a linha 39 faz com que o L-BFGS atualize a imagem e o rótulo fictícios com base nessa função objetivo. Ao repetir esse processo por várias iterações, a entrada inicialmente aleatória tende a evoluir para uma amostra capaz de reproduzir os gradientes observados, o que constitui a reconstrução do ataque. Por fim, na linha 46, a implementação recupera o rótulo inferido a partir do vetor otimizado. Assim, ao final do processo, o ataque produz não apenas uma imagem reconstruída, mas também uma estimativa do rótulo correspondente.

A Figura 1.8 apresenta dois exemplos de execução do ataque DLG, um para uma amostra da classe “dinossauro” e outro para uma amostra da classe “gado”. Em ambos os



(a) Reconstrução de uma amostra da classe “dinossauro”.



(b) Reconstrução de uma amostra da classe “gado”.

Figura 1.8. Evolução do ataque DLG ao longo das iterações. O gráfico superior mostra a redução da função de perda de correspondência de gradientes, enquanto as imagens inferiores ilustram a progressão da reconstrução, partindo de ruído aleatório até uma amostra visualmente próxima da imagem original.

casos, observa-se a redução progressiva da função de perda associada à correspondência entre gradientes reais e gradientes fictícios ao longo das iterações, enquanto é mostrada a evolução visual da reconstrução, iniciando-se a partir de ruído aleatório e convergindo gradualmente para uma imagem com estrutura semântica próxima da amostra original.

Diferentemente do DLG, em que o adversário é o servidor agregador, na reconstrução com GAN o atacante é modelado como um cliente participante curioso. O Código fonte sompleto está disponível no GitLab[9]. Essa implementação se baseou exatamente na descrição do trabalho [Zhu et al., 2019].

A lógica principal do atacante é implementada na classe `AdversaryClient`, implementada em `client_app.py`[10]. Essa classe representa o cliente curioso res-

[9]Disponível em `https://gitlab.com/LCC/fl-gan-reconstruction-attack`. Acessado em 21/04/2026

[10]Disponível em `https://gitlab.com/LCC/fl-gan-reconstruction-attack/-/blob/main/gan_attack_flower/client_app.py`. Acessado em 21/04/2026

ponsável por treinar o gerador, sintetizar amostras, inseri-las em seu treinamento local e devolver ao servidor uma atualização contaminada por esse processo.

O comportamento malicioso começa no método `fit()`. Nesse método o cliente malicioso usa o modelo global recebido como o discriminador, posteriormente usado no trinamento do gerador. Em outras palavras, o modelo global funciona como a fonte de informação que incorpora conhecimento extraído dos dados privados da vítima.

Ainda no método `fit()`, o cliente chama o método `train_generator(...)`. Nesse método, o modelo global atual é usado como discriminador, e o gerador é otimizado para produzir amostras que o classificador reconheça como pertencentes à `target_class`. Em termos práticos, o atacante usa o modelo global como um oráculo diferenciável que contém informação sobre a classe privada da vítima. Quanto melhor o gerador aprender a explorar esse oráculo, mais próximas da classe-alvo tenderão a ser as imagens sintetizadas.

Posteriormente, o cliente gera um conjunto de amostras sintéticas por meio de `generate_samples(...)` e cria um vetor de rótulos artificiais usando `FAKE_CLASS_INDEX`. Esse é um dos pontos mais característicos do ataque; as amostras produzidas pela GAN tentam se parecer com a classe privada da vítima, mas são propositalmente rotuladas com uma classe artificial. Esse mecanismo implementa o engano sobre o modelo global.

O cliente, então, começa a executar o treinamento local chamando o método `train_classifier(...)` com os tensores `injected_x` e `injected_y`, gerador por meio do método `generate_samples(...)`. Essa etapa é onde as amostras geradas pela GAN deixam de ser apenas um artefato de inspeção visual e passam a interferir diretamente na atualização local devolvida ao servidor. Em outras palavras, o atacante injeta amostras sintéticas de aparência próxima à classe-alvo, mas com uma classe artificial, e força o classificador a ajustar suas fronteiras de decisão em resposta a esse estímulo. É esse ciclo de realimentação entre geração e treinamento federado que torna o ataque eficaz ao longo das rodadas.

Por fim, o cliente retorna ao servidor os parâmetros atualizados do classificador, o tamanho efetivo do conjunto de treinamento utilizado e métricas adicionais, entre elas a perda do treinamento local e a perda do gerador. O ponto importante é que o servidor recebe uma atualização aparentemente legítima, mas essa atualização já foi influenciada pela injeção adversarial de amostras sintéticas.

Na implementação, foi desenvolvido um arquivo chamado `attack.py`[11] para auxiliar no ataque. As funções principais do arquivo `attack.py` são apresentadas no Código 1.6. As funções apresentadas implementam o treinamento do gerador contra o modelo global e a geração das amostras sintéticas usadas pelo atacante.

```
1 def train_generator(
2     generator: Generator,
3     discriminator: nn.Module,
4     target_class: int,
5     latent_dim: int,
```

[11] Diponível em `https://gitlab.com/LCC/fl-gan-reconstruction-attack/-/blob/main/gan_attack_flower/attack.py`. Acessado em 21/04/2026.

```
6       steps: int,
7       batch_size: int,
8       lr: float,
9       device: torch.device,
10  ) -> float:
11      discriminator.eval()
12      for p in discriminator.parameters():
13          p.requires_grad_(False)
14
15      generator.train()
16      optimizer = torch.optim.SGD(generator.parameters(), lr=lr, momentum
        =0.0)
17      criterion = nn.NLLLoss()
18
19      target = torch.full((batch_size,), target_class, dtype=torch.long,
        device=device)
20
21      last_loss = 0.0
22      for _ in range(steps):
23          optimizer.zero_grad()
24          z = torch.randn(batch_size, latent_dim, device=device)
25          fake = generator(z)
26          log_probs = discriminator(fake)
27          loss = criterion(log_probs, target)
28          loss.backward()
29          optimizer.step()
30          last_loss = float(loss.item())
31
32      for p in discriminator.parameters():
33          p.requires_grad_(True)
34
35      return last_loss
36
37  @torch.no_grad()
38  def generate_samples(
39      generator: Generator,
40      num_samples: int,
41      latent_dim: int,
42      device: torch.device,
43  ) -> torch.Tensor:
44      generator.eval()
45      z = torch.randn(num_samples, latent_dim, device=device)
46      return generator(z)
```

Código Fonte 1.6. Funções relevantes do arquivo `attack.py`

Na função `train_generator()`, o modelo global é usado no modo avaliação para atuar apenas como fonte de sinal para o ataque, enquanto o gerador passa a ser o único componente efetivamente otimizado. Em seguida, define-se como objetivo fazer com que as amostras sintéticas produzidas pelo gerador sejam classificadas pelo modelo global como pertencentes à `target_class`, que corresponde à classe privada da vítima. O núcleo do ataque ocorre no laço iterativo (linhas 22), em que os ruídos aleatórios são convertidos em imagens sintéticas; essas imagens são submetidas ao modelo global e a perda resultante é retropropagada apenas para atualizar o gerador. Dessa forma, a cada

passo de otimização, o gerador é ajustado para produzir amostras cada vez mais compatíveis com a classe-alvo segundo o conhecimento do modelo global. Ao final, os gradientes do modelo global (linhas 32 e 33) são reabilitados para que ele volte a participar normalmente do treinamento federado nas etapas seguintes.

A função `generate_samples()` é mais simples, mas também é essencial para o ataque. Nas linhas 45 a 47, o gerador é colocado em modo de avaliação, um conjunto de ruídos aleatórios é amostrado e, na linha 47, esses vetores são convertidos em imagens sintéticas. Em conjunto, essas funções mostram que o ataque GAN em Aprendizado Federado não consiste apenas em treinar um gerador isolado. O ponto principal é a interação entre três elementos: (i) o modelo global recebido pelo cliente curioso, (ii) o treinamento do gerador para explorar a informação contida nesse modelo e (iii) a injeção das amostras geradas no treinamento local do próprio atacante. Essa combinação é que transforma o cliente curioso em um adversário capaz de reconstruir progressivamente conteúdo privado da vítima.

A Figura 1.9 compara amostras originais com as respectivas reconstruções obtidas pelo ataque. De modo geral, observa-se que as imagens reconstruídas preservam a estrutura global e os traços mais característicos da classe, embora apresentem maior desfoque e menor nitidez em relação às amostras originais. Ainda assim, a similaridade entre as duas linhas indica que o ataque foi capaz de recuperar informação visual relevante sobre o conteúdo privado, mesmo sem acesso direto aos dados reais. Esse resultado reforça que, em FL, as informações exploradas pelo atacante podem ser suficientes para reconstruir amostras com alto grau de correspondência estrutural com os dados originais.

1.6. Discussão, Tendências e Desafios de Pesquisa

O Aprendizado Federado tem sido aplicado em diferentes cenários que envolvem dados distribuídos e sensíveis, nos quais a centralização das informações é limitada por requisitos de privacidade, regulamentação ou infraestrutura. Nessa condição, sua adoção evidencia tanto o potencial do treinamento colaborativo quanto a complexidade associada à sua utilização em ambientes reais. Diferentes domínios de aplicação impõem requisitos específicos ao uso do Aprendizado Federado, influenciando diretamente os desafios enfrentados na sua implementação e as soluções investigadas na literatura. Inicialmente, apresentam-se aplicações críticas e requisitos de segurança em Aprendizado Federado, em seguida são discutidos os principais desafios em aberto e, por fim, são descritos projetos e linhas de pesquisa voltados à mitigação dessas limitações.

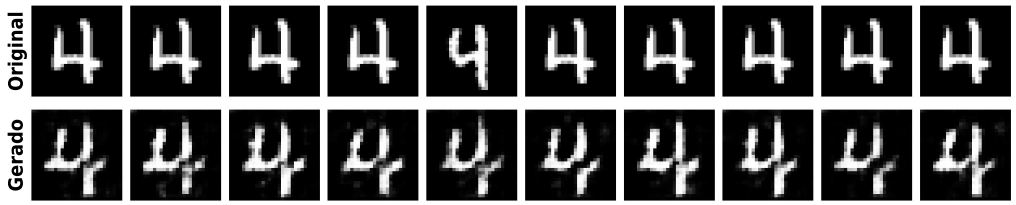


Figura 1.9. Comparação entre amostras originais e reconstruções obtidas pelo ataque. A linha superior apresenta exemplos de imagens originais, enquanto a linha inferior mostra as respectivas imagens reconstruídas, permitindo avaliar visualmente o grau de preservação da estrutura da classe alvo.

1.6.1. Aplicações críticas e requisitos de segurança em Aprendizado Federado

O Aprendizado Federado tem sido empregado em aplicações nas quais a proteção dos dados dos usuários e a conformidade com regulamentações vigentes são requisitos centrais [Wen et al., 2023]. Nesse contexto, diferentes domínios de aplicação impõem requisitos específicos para a operação segura e eficiente desse paradigma, condicionando as técnicas a serem adotadas. Na área da saúde, o Aprendizado Federado permite treinar modelos a partir de dados distribuídos entre instituições, ampliando o espaço amostral e favorecendo a generalização em tarefas como a classificação de imagens médicas [Rieke et al., 2020]. Esse domínio é marcado pela alta sensibilidade dos dados e por regulamentações rigorosas, de modo que aplicações de Aprendizado Federado em saúde demandam mecanismos de governança capazes de definir responsabilidades, regras de acesso, procedimentos de compartilhamento e critérios de uso dos modelos resultantes [Eden et al., 2025]. Nessa configuração, garantir confidencialidade, integridade das atualizações e confiabilidade do processo de treinamento torna-se essencial para evitar vazamentos de informação ou manipulações que comprometam os resultados obtidos [Nguyen et al., 2022].

No setor financeiro, aplicações como detecção de fraudes e análise de risco de crédito se beneficiam da colaboração entre instituições sem compartilhamento direto de dados sensíveis. Trata-se de um domínio altamente regulado, sujeito a normas internas e legislações de proteção de dados que restringem o trânsito de informações pessoais e transacionais [Kennedy et al., 2025]. Além da privacidade, robustez e confiabilidade dos modelos são requisitos fundamentais, pois decisões derivadas desses sistemas impactam diretamente indivíduos e organizações, exigindo garantias de integridade das contribuições e resistência a comportamentos adversariais.

Em cenários de cidades inteligentes e IoT, o Aprendizado Federado permite explorar dados provenientes de sensores e dispositivos distribuídos sem centralizar todas as medições em um único ponto. Esses ambientes são caracterizados por alta heterogeneidade, tanto em termos de distribuição dos dados quanto de capacidade computacional e de comunicação dos dispositivos participantes [Pandya et al., 2023]. Nessas condições, mecanismos flexíveis e adaptativos tornam-se necessários para lidar com limitações de processamento, variações de conectividade e diferentes níveis de desempenho, ao mesmo tempo em que requisitos de escalabilidade, eficiência de comunicação e tolerância à variabilidade dos dispositivos são determinantes para viabilizar o uso em larga escala.

De forma geral, diferentes domínios de aplicação atribuem pesos distintos a propriedades como segurança, privacidade, robustez, escalabilidade e eficiência operacional, influenciando diretamente o desenho dos sistemas de Aprendizado Federado [Pandya et al., 2023]. Essa diversidade de requisitos evidencia a complexidade de aplicar o paradigma em ambientes reais e prepara o terreno para os desafios em aberto discutidos na subseção seguinte. Os requisitos das aplicações críticas em diferentes domínios de aplicação do Aprendizado Federado são exemplificados na Tabela 1.2.

1.6.2. Desafios em Aberto

O paradigma de Aprendizado Federado pode mitigar a exposição direta de dados locais, oferecer maior flexibilidade para atender a requisitos regulatórios de proteção de

Tabela 1.2. Exemplos de domínios de aplicação de Aprendizado Federado e requisitos associados.

Domínio	Principais requisitos	Impacto dos requisitos
Saúde	Privacidade forte, conformidade regulatória, governança, integridade das atualizações, robustez a ataques	Restringe o acesso a dados brutos, exige mecanismos de proteção contra vazamento e manipulação, e condiciona o uso clínico dos modelos a critérios de confiança e auditabilidade.
Setor financeiro	Privacidade de dados sensíveis, conformidade com normas e leis, robustez, confiabilidade decisória	Impõe alta exigência de segurança e resiliência a ataques, pois erros ou manipulações podem afetar diretamente decisões de crédito, detecção de fraudes e riscos institucionais.
Cidades inteligentes e IoT	Escalabilidade, eficiência de comunicação, tolerância a heterogeneidade, privacidade, robustez	Exige protocolos leves e adaptativos que operem com dispositivos limitados e redes instáveis, mantendo proteção de dados e desempenho aceitável em grande escala.

dados e viabilizar a colaboração entre diferentes instituições no treinamento de modelos. Apesar dessas vantagens, a adoção prática ainda enfrenta desafios importantes, pois a necessidade de garantir segurança, privacidade, confiabilidade e integridade de dados e modelos impõe restrições significativas ao uso em ambientes reais [Zhao et al., 2025]. Esses obstáculos extrapolam o nível algorítmico e envolvem também limitações de infraestrutura, heterogeneidade entre participantes e restrições operacionais típicas de sistemas distribuídos [Kairouz e McMahan, 2021].

Um desafio central em Aprendizado Federado é a heterogeneidade inerente a ambientes descentralizados. Essa heterogeneidade manifesta-se, principalmente, na forma de heterogeneidade estatística dos dados, frequentemente associada ao cenário não independente e não identicamente distribuído (*non-IID*), e na heterogeneidade de recursos e condições dos próprios clientes. A heterogeneidade estatística [Lu et al., 2024] pode comprometer a convergência e a capacidade de generalização do modelo global, enquanto as diferenças em capacidade computacional, disponibilidade de recursos e qualidade de conexão de rede introduzem desequilíbrios no treinamento, como atrasos de atualização e ociosidade decorrente de clientes lentos [Ye et al., 2023]. Embora essas variações possam enriquecer a diversidade dos dados, a combinação de distribuição *non-IID* e heterogeneidade de sistema tende a dificultar a convergência, aumentar o custo de comunicação e reduzir a qualidade final do modelo em cenários práticos [Lu et al., 2024, Ye et al., 2023].

Outro desafio amplamente discutido refere-se ao equilíbrio entre privacidade, custo e desempenho em Aprendizado Federado. A redução da exposição direta dos dados dos clientes é uma das principais motivações para adoção do paradigma federado, o que torna essencial a proteção contra vazamentos ou inferência de informações sensíveis [Zhao et al., 2025]. Nesse contexto, técnicas como privacidade diferencial e *secure aggregation* são empregadas para mitigar riscos de exposição dos dados locais [Geyer et al., 2017, Bonawitz et al., 2017], mas introduzem um compromisso entre privacidade, utilidade e custos computacionais. A privacidade diferencial adiciona ruído aos dados ou às atualizações, podendo deteriorar a acurácia e a eficiência do modelo global [Geyer

et al., 2017], enquanto a *secure aggregation* eleva o custo computacional e de comunicação e restringe o acesso a contribuições individuais dos clientes [Bonawitz et al., 2017]. Como consequência, essas camadas de proteção reduzem a visibilidade sobre o processo de treinamento, dificultam a análise de falhas, vieses e comportamentos maliciosos e podem levar a modelos globais com desempenho degradado, vieses indesejados ou comportamentos inconsistentes [Zhao et al., 2025].

A literatura também identifica como desafios atuais e relevantes ataques de envenenamento (*poisoning*), ataques Bizantinos e ataques *backdoor*, nos quais clientes maliciosos enviam atualizações manipuladas com o objetivo de degradar o modelo global ou induzir comportamentos específicos [Zhao et al., 2025,Chen et al., 2022]. Como resposta, técnicas de agregação robusta têm sido propostas para reduzir a influência de atualizações inconsistentes ou maliciosas no treinamento federado [Pillutla et al., 2022]. Ainda assim, essas abordagens apresentam limitações importantes em cenários com dados *non-IID*, nos quais a variabilidade legítima entre clientes pode mimetizar comportamentos adversariais, tornando mais difícil distinguir contribuições benignas de ações maliciosas [Lu et al., 2024, Pillutla et al., 2022].

Além desses aspectos, o Aprendizado Federado enfrenta desafios de escalabilidade e restrições operacionais dos sistemas distribuídos. A diversidade de características dos clientes, como capacidade de processamento, qualidade de conexão de rede e limitações de memória, impacta diretamente a eficiência do treinamento colaborativo e pode restringir o uso de Aprendizado Federado em cenários com requisitos rígidos de qualidade de serviço (QoS) [Ye et al., 2023]. Em ambientes reais, a heterogeneidade de desempenho entre clientes dificulta a sincronização das atualizações e a participação equilibrada, resultando em atrasos na agregação, aumento do tempo total de treinamento e maior ociosidade do sistema devido a participantes mais lentos, além de problemas de conectividade que levam à perda ou reenvio de atualizações [Ye et al., 2023]. Nessa perspectiva, o desenvolvimento de técnicas que ofereçam maior escalabilidade e flexibilidade nos requisitos impostos aos clientes permanece como um desafio crucial para a adoção do Aprendizado Federado em larga escala.

Os desafios de Aprendizado Federado precisam ser avaliados de forma integrada.Em particular, abordagens que fortalecem privacidade, robustez ou eficiência podem, simultaneamente, afetar negativamente o desempenho, a visibilidade sobre o processo de treinamento ou a escalabilidade, evidenciando a necessidade de uma análise conjunta entre objetivos e restrições [Kairouz e McMahan, 2021]. Embora muitas propostas da literatura ainda tratem problemas de forma isolada, observa-se um movimento crescente de modularização das soluções, permitindo sua adaptação e composição em diferentes arquiteturas e cenários de aplicação; essa tendência aponta para avanços não apenas na mitigação de reveses específicos, mas também na capacidade de integrar mecanismos especializados de maneira flexível e ajustada a cada cenário [Kairouz e McMahan, 2021]. A Tabela 1.3 elenca os principais desafios e propostas de mitigação.

1.6.3. Projetos de Pesquisa

Diante dos desafios de FL, diversas propostas na literatura têm buscado mitigar as suas limitações por meio de abordagens voltadas à privacidade, robustez e eficiência

Tabela 1.3. Principais desafios em Aprendizado Federado, dimensões afetadas e formas de mitigação.

Desafio	Dimensão	Mitigação
Heterogeneidade *non-IID* e impacto na convergência	Dados	Ajustes de protocolo e algoritmos de otimização/agregação adaptados a cenários *non-IID* para reduzir desequilíbrios entre contribuições [Lu et al., 2024, Pillutla et al., 2022].
Heterogeneidade de recursos dos clientes	Clientes	Estratégias assíncronas ou parciais e requisitos mais flexíveis de participação para acomodar diferenças de capacidade e conectividade [Ye et al., 2023].
Equilíbrio entre privacidade, custo e desempenho	Dados/ Agregação	Uso de privacidade diferencial e *secure aggregation* com calibração de ruído e de protocolo para limitar vazamento mantendo utilidade aceitável [Geyer et al., 2017, Bonawitz et al., 2017, Lyu, 2020].
Ataques de envenenamento, Bizantinos e *backdoor*	Dados/ Agregação	Agregação robusta, filtragem de clientes suspeitos e técnicas específicas de detecção e remoção de *backdoors* [Pillutla et al., 2022, Chen et al., 2022, Li et al., 2022b].
Escalabilidade e restrições de sistemas distribuídos	Servidor/ Seleção	Controle do número de clientes por rodada, técnicas de comunicação eficientes e algoritmos tolerantes a falhas e atrasos [Ye et al., 2023, Kairouz e McMahan, 2021].
Interdependência entre privacidade, robustez, desempenho e escalabilidade	Sistema	Desenho modular de defesas e análise conjunta de *trade-offs* para combinar mecanismos especializados de forma flexível [Kairouz e McMahan, 2021].

dos sistemas [Kairouz e McMahan, 2021]. Essas iniciativas evidenciam a natureza multiobjetivo do problema em ambientes distribuídos, indicando que soluções eficazes devem considerar as particularidades de cada cenário de aplicação e seus respectivos requisitos.

Um avanço recente nesse contexto é o desenvolvimento de metodologias de avaliação mais estruturadas, voltadas à análise sistemática de ataques e mecanismos de defesa [Han et al., 2024]. A criação de *benchmarks* padronizados tem aumentado a comparabilidade entre diferentes abordagens, permitindo avaliações mais consistentes dos métodos propostos e favorecendo a reprodutibilidade dos experimentos, o que facilita a validação de resultados e a comparação entre soluções baseadas em estratégias distintas. Paralelamente, o desenvolvimento de ferramentas e plataformas específicas tem desempenhado papel central na viabilização prática do modelo federado.

Entre essas iniciativas, destaca-se o projeto OpenMined, responsável pela biblioteca PySyft, destinada à implementação de Aprendizado Federado com ênfase em privacidade [Nguyen et al., 2024]. A PySyft permite treinar modelos sem acesso direto aos dados locais, incorporando mecanismos que viabilizam o compartilhamento seguro de informações entre entidades participantes. Outra iniciativa relevante é o Flower, um *framework* projetado para ser simples e flexível, permitindo a implementação de sistemas federados de forma agnóstica à biblioteca de aprendizado de máquina subjacente e possibilitando a orquestração de ambientes federados simulados e aplicações reais [Nguyen et al., 2024].

Além dessas bibliotecas, o Aprendizado Federado já está presente em aplicações comerciais consolidadas. Um exemplo notório é o teclado virtual do Google (Gboard), que utiliza Aprendizado Federado para aprimorar a predição de texto diretamente nos dispositivos dos usuários, treinando modelos locais a partir dos padrões de escrita individuais e compartilhando apenas atualizações de modelo com o servidor, o que contribui para pre-

servar a privacidade das informações sensíveis [Hard et al., 2018]. Esse tipo de aplicação ilustra a viabilidade do paradigma federado em larga escala quando há alinhamento entre requisitos de privacidade e capacidade de infraestrutura.

De forma geral, observa-se que muitas soluções propostas ainda se concentram em desafios específicos e são desenvolvidas como mecanismos especializados voltados a problemas particulares. Nesse cenário, ganham destaque abordagens modularizadas, que promovem maior flexibilidade de adoção e integração em diferentes arquiteturas e fluxos de treinamento [Kairouz e McMahan, 2021]. Esse movimento aponta para uma linha de pesquisa orientada à composição de mecanismos especializados, buscando equilibrar, de maneira adaptável, múltiplos requisitos do sistema em cenários reais de FL.

1.7. Considerações Finais

O Aprendizado Federado (*Federated Learning* – FL) é um paradigma promissor para treinamento colaborativo de modelos em ambientes com dados distribuídos e sensíveis, oferecendo uma alternativa à centralização tradicional ao reduzir a exposição direta de informações e facilitar a conformidade com requisitos regulatórios. Ao longo deste capítulo, foi discutido como esse paradigma também conta com uma superfície de ataque mais ampla, em que vulnerabilidades ligadas a envenenamento de modelo, ataques Bizantinos, *backdoors* e diferentes formas de quebra de privacidade colocam em risco tanto o desempenho quanto a confidencialidade dos modelos treinados. A análise dessas ameaças evidencia que a segurança em Aprendizado Federado não é um problema pontual, mas um componente intrínseco ao projeto do sistema.

No eixo da privacidade, foram discutidos ataques de inferência de participação, inferência de atributos, inferência de origem e reconstrução de conteúdo, todos explorando o fato de que atualizações de modelo, gradientes e representações internas carregam sinais sobre os dados locais. Estratégias como privacidade diferencial, criptografia homomórfica, agregação segura e computação multipartidária têm mostrado potencial para limitar esse vazamento, ainda que introduzam compromissos importantes entre proteção de privacidade, custo computacional, custo de comunicação e acurácia do modelo. Em cenários práticos, o desafio deixa de ser apenas "proteger ao máximo" e passa a envolver o ajuste fino de parâmetros e mecanismos para atingir níveis de proteção compatíveis com as exigências regulatórias e os requisitos da aplicação.

No eixo da robustez, foram caracterizados ataques de envenenamento de dados, envenenamento de modelo, ataques Bizantinos e ataques direcionados (*backdoor*), incluindo variantes por substituição de modelo, casos raros e formulações baseadas em otimização ou aprendizado por reforço. Técnicas de agregação robusta, como Krum, *Trimmed Mean*, Bulyan e RFA, bem como métodos de filtragem de clientes suspeitos foram apresentadas como defesas centrais contra ataques não-direcionados. Para ataques direcionados, defesas especializadas de detecção e eliminação, baseadas em assinaturas espectrais, agrupamento de ativações, *fine-pruning*, distilação de atenção, reparo por conectividade de modos e aprendizagem anti-*backdoor*, exemplificam a necessidade de mecanismos que operem sobre a estrutura interna do modelo e não apenas sobre estatísticas agregadas.

Em termos de desafios sistêmicos, foram destacados os efeitos da heterogeneidade estatística de dados (dados *non-IID*) e da heterogeneidade de recursos entre clientes, que

impactam a convergência, a estabilidade e o custo de treinamento. Esses fatores interagem de forma não trivial com os mecanismos de segurança, já que agregadores robustos podem confundir variabilidade legítima com comportamento malicioso, mecanismos de proteção de privacidade reduzem a visibilidade sobre contribuições individuais, e restrições de dispositivos e comunicação limitam a complexidade das defesas viáveis em produção. Como consequência, o projeto de sistemas de Aprendizado Federado em ambientes reais exige considerar simultaneamente dimensões de dados, clientes, seleção de participantes, agregação e papel do servidor, com atenção às relações de compromisso entre privacidade, robustez, desempenho e escalabilidade.

As aplicações críticas discutidas, como saúde, finanças, cidades inteligentes, IoT e serviços móveis, reforçam que não existe uma configuração única de segurança adequada a todos os cenários. Em alguns domínios, propriedades como confidencialidade, governança e auditabilidade são prioritárias, enquanto em outros a viabilidade do Aprendizado Federado depende principalmente de requisitos de Qualidade de Serviço (*Quality of Service* – QoS), eficiência e tolerância à heterogeneidade. O levantamento de projetos e *frameworks*, como PySyft e Flower, e o exemplo de uso em larga escala no Gboard da Google mostram que já existem ferramentas práticas para integrar mecanismos de segurança ao ciclo de desenvolvimento, experimentação e implantação de soluções federadas, aproximando a pesquisa acadêmica de contextos operacionais.

Observa-se que a maior parte das contribuições atuais ainda aborda problemas específicos de forma isolada, mas há um movimento crescente em direção a arquiteturas modulares que permitam combinar, de forma configurável, mecanismos de privacidade, agregação robusta, monitoramento e detecção de anomalias. Essa tendência indica que os próximos avanços relevantes em segurança para Aprendizado Federado deverão partir menos de técnicas únicas e mais de composições ajustadas, capazes de adaptar o nível de proteção às necessidades do domínio de aplicação, aos riscos aceitáveis e às limitações de infraestrutura presentes em ambientes reais.

Referências

[Agarwal et al., 2018] Agarwal, N., Suresh, A. T., Yu, F. X. X., Kumar, S. e McMahan, B. (2018). cpsgd: Communication-efficient and Differentially-private Distributed Sgd. *Advances in neural information processing systems*, 31.

[Blanchard et al., 2017] Blanchard, P., El Mhamdi, E. M., Guerraoui, R. e Stainer, J. (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems*, 30.

[Bochie et al., 2021] Bochie, K., Sammarco, M., Detyniecki, M. e Campista, M. E. M. (2021). Análise do aprendizado federado em redes móveis. Em *SBRC*, p. 71–84. SBC.

[Bonawitz et al., 2017] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A. e Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. Em *Proceedings of the 2017 ACM SIGSAC*, CCS '17, p. 1175–1191, New York, NY, USA. ACM.

[Brendan McMahan et al., 2017] Brendan McMahan, H., Moore, E., Ramage, D., Hampson, S. e Agüera y Arcas, B. (2017). Communication-efficient learning of deep networks from decentralized data. Em *Proceedings of the 20th AISTATS*, volume 54.

[Cao et al., 2020] Cao, X., Fang, M., Liu, J. e Gong, N. Z. (2020). fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*.

[Cao et al., 2021] Cao, X., Jia, J. e Gong, N. Z. (2021). Provably secure federated learning against malicious clients. Em *Proceedings of the AAAI conference on artificial intelligence*, volume 35, p. 6885–6893.

[Chen et al., 2018] Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I. e Srivastava, B. (2018). Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*.

[Chen et al., 2022] Chen, Y., Gui, Y., Lin, H., Gan, W. e Wu, Y. (2022). Federated learning attacks and defenses: a survey. Em *2022 IEEE international conference on big data (big data)*, p. 4256–4265. IEEE.

[Cunha Neto et al., 2023] Cunha Neto, H. N., Hribar, J., Dusparic, I., Mattos, D. M. F. e Fernandes, N. C. (2023). A survey on securing federated learning: Analysis of applications, attacks, challenges, and trends. *IEEE Access*, 11:41928–41953.

[de Assis et al., 2023] de Assis, F. M., Macedo, E. L. e de Moraes, L. F. (2023). Aplicaçao de criptografia homomórfica na mineraçao de dados em fluxos de roteadores de borda na internet. Em *SBSeg*, p. 273–278. SBC.

[de Assis et al., 2024] de Assis, F. M., Macedo, E. L. e de Moraes, L. F. (2024). Métodos para criptografia homomórfica na mineraçao de dados aplicados em fluxos de roteadores de borda na internet. Em *WGRS*, p. 70–83. SBC.

[Diana et al., 2025] Diana, F., Marfoq, O., Xu, C., Neglia, G., Giroire, F. e Thomas, E. (2025). Attribute inference attacks for federated regression tasks. Em *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, p. 16271–16279.

[Dwork e Roth, 2014] Dwork, C. e Roth, A. (2014). The algorithmic foundations of differential privacy. *Foundations and trends® in theoretical computer science*, 9(3-4):211–487.

[Eden et al., 2025] Eden, R., Chukwudi, I., Bain, C., Barbieri, S., Callaway, L., de Jersey, S., George, Y., Gorse, A.-D., Lawley, M., Marendy, P. et al. (2025). A scoping review of the governance of federated learning in healthcare. *npj Digital Medicine*, 8(1):427.

[Fang et al., 2020] Fang, M., Cao, X., Jia, J. e Gong, N. Z. (2020). Local model poisoning attacks to byzantine-robust federated learning. Em *29th USENIX*, p. 1623–1640.

[Fang et al., 2024] Fang, M., Zhang, Z., Hairi, Khanduri, P., Liu, J., Lu, S., Liu, Y. e Gong, N. (2024). Byzantine-robust decentralized federated learning. Em *Proceedings of the 2024 on ACM SIGSAC*, p. 2874–2888.

[Fu et al., 2022] Fu, C., Zhang, X., Ji, S., Chen, J., Wu, J., Guo, S., Zhou, J., Liu, A. X. e Wang, T. (2022). Label inference attacks against vertical federated learning. Em *31st USENIX security symposium (USENIX Security 22)*, p. 1397–1414.

[Fung et al., 2018] Fung, C., Yoon, C. J. e Beschastnikh, I. (2018). Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*.

[Geyer et al., 2017] Geyer, R. C., Klein, T. e Nabi, M. (2017). Differentially private federated learning: a client level perspective. *arXiv preprint arXiv:1712.07557*.

[Goldblum et al., 2022] Goldblum, M., Tsipras, D., Xie, C., Chen, X., Schwarzschild, A., Song, D., Madry, A., Li, B. e Goldstein, T. (2022). Dataset security for machine learning: data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, p. 1–1.

[Guerraoui et al., 2018] Guerraoui, R., Rouault, S. et al. (2018). The hidden vulnerability of distributed learning in byzantium. Em *International conference on machine learning*, p. 3521–3530. PMLR.

[Han et al., 2024] Han, S., Buyukates, B., Hu, Z., Jin, H., Jin, W., Sun, L., Wang, X., Wu, W., Xie, C., Yao, Y. et al. (2024). Fedsecurity: a benchmark for attacks and defenses in federated learning and federated llms. Em *30th ACM SIGKDD*, p. 5070–5081.

[Hard et al., 2018] Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C. e Ramage, D. (2018). Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

[Hitaj et al., 2017] Hitaj, B., Ateniese, G. e Perez-Cruz, F. (2017). Deep models under the gan: Information leakage from collaborative deep learning. Em *2017 ACM SIGSAC*, CCS '17, p. 603–618, New York, NY, USA. ACM.

[Hu et al., 2023] Hu, H., Zhang, X., Salcic, Z., Sun, L., Choo, K.-K. R. e Dobbie, G. (2023). Source inference attacks: Beyond membership inference attacks in federated learning. *IEEE Transactions on Dependable and Secure Computing*, 21(4):3012–3029.

[Jayaraman e Evans, 2022] Jayaraman, B. e Evans, D. (2022). Are attribute inference attacks just imputation? Em *2022 ACM SIGSAC*, p. 1569–1582.

[Ji et al., 2025] Ji, A., Bandyopadhyay, B., Song, C., Krishnaswami, N., Vashisht, P., Smiroldo, R., Litton, I., Mahinder, S., Chitnis, M. e Hill, A. W. (2025). Private federated learning in real world application – a case study.

[Kairouz e McMahan, 2021] Kairouz, P. e McMahan, H. B. (2021). Advances and open problems in federated learning. *Foundations and trends in ML*, 14(1-2):1–210.

[Kennedy et al., 2025] Kennedy, C. H., Hilal, A. e Momeni, M. (2025). The role of federated learning in improving financial security: a survey. Em *2025 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, p. 1–8. IEEE.

[Li et al., 2022a] Li, H., Sun, X. e Zheng, Z. (2022a). Learning to attack federated learning: a model-based reinforcement learning attack framework. *Advances in Neural Information Processing Systems*, 35:35007–35020.

[Li et al., 2023] Li, S., Ngai, E. C.-H. e Voigt, T. (2023). An experimental study of byzantine-robust aggregation schemes in federated learning. *IEEE Transactions on Big Data*, 10(6):975–988.

[Li et al., 2022b] Li, Y., Jiang, Y., Li, Z. e Xia, S.-T. (2022b). Backdoor learning: a survey. *IEEE transactions on neural networks and learning systems*, 35(1):5–22.

[Li et al., 2021a] Li, Y., Lyu, X., Koren, N., Lyu, L., Li, B. e Ma, X. (2021a). Anti-backdoor learning: Training clean models on poisoned data. *Advances in Neural Information Processing Systems*, 34:14900–14912.

[Li et al., 2021b] Li, Y., Lyu, X., Koren, N., Lyu, L., Li, B. e Ma, X. (2021b). Neural attention distillation: Erasing backdoor triggers from deep neural networks. *arXiv preprint arXiv:2101.05930*.

[Lim et al., 2020] Lim, W. Y. B., Luong, N. C., Hoang, D. T., Jiao, Y., Liang, Y. C., Yang, Q., Niyato, D. e Miao, C. (2020). Federated learning in mobile edge networks: a comprehensive survey. *IEEE Communications Surveys Tutorials*.

[Liu e Nocedal, 1989] Liu, D. C. e Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1–3):503–528.

[Liu et al., 2018] Liu, K., Dolan-Gavitt, B. e Garg, S. (2018). Fine-pruning: Defending against backdooring attacks on deep neural networks. Em *International symposium on research in attacks, intrusions, and defenses*, p. 273–294. Springer.

[Liu et al., 2022] Liu, P., Xu, X. e Wang, W. (2022). Threats, attacks and defenses to federated learning: Issues, taxonomy and perspectives. *Cybersecurity*, 5(1):1–19.

[Lu et al., 2024] Lu, Z., Pan, H., Dai, Y., Si, X. e Zhang, Y. (2024). Federated learning with non-iid data: a survey. *IEEE Internet of Things Journal*, 11(11):19188–19209.

[Lyu, 2020] Lyu, L. (2020). Lightweight crypto-assisted distributed differential privacy for privacy-preserving distributed learning. Em *2020 IJCNN*, p. 1–8. IEEE.

[Lyu et al., 2022] Lyu, L., Yu, H., Ma, X., Chen, C., Sun, L., Zhao, J., Yang, Q. e Yu, P. S. (2022). Privacy and robustness in federated learning: Attacks and defenses. *IEEE transactions on neural networks and learning systems*, 35(7):8726–8746.

[Lyu et al., 2020] Lyu, L., Yu, H. e Yang, Q. (2020). Threats to federated learning: a survey. *arXiv preprint arXiv:2003.02133*.

[McMahan et al., 2016] McMahan, H. B., Moore, E., Ramage, D. e y Arcas, B. A. (2016). Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2(2):15–18.

[McMahan et al., 2018] McMahan, H. B., Ramage, D., Talwar, K. e Zhang, L. (2018). Learning differentially private recurrent language models. Em *International Conference on Learning Representations*.

[Melis et al., 2019] Melis, L., Song, C., De Cristofaro, E. e Shmatikov, V. (2019). Exploiting unintended feature leakage in collaborative learning. Em *2019 IEEE symposium on security and privacy (SP)*, p. 691–706. IEEE.

[Nasr et al., 2019] Nasr, M., Shokri, R. e Houmansadr, A. (2019). Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. Em *2019 IEEE symposium on security and privacy (SP)*, p. 739–753. IEEE.

[Nguyen et al., 2022] Nguyen, D. C., Pham, Q.-V., Pathirana, P. N., Ding, M., Seneviratne, A., Lin, Z., Dobre, O. e Hwang, W.-J. (2022). Federated learning for smart healthcare: A survey. *ACM Comput. Surv.*, 55(3).

[Nguyen et al., 2024] Nguyen, G., Sainz-Pardo Diaz, J., Calatrava, A., Berberi, L., Lytvyn, O., Kozlov, V., Tran, V., Moltó, G. e Lopez Garcia, A. (2024). Landscape of machine learning evolution: Privacy-preserving federated learning frameworks and tools: G. nguyen et al. *Artificial Intelligence Review*, 58(2):51.

[Nowroozi et al., 2025] Nowroozi, E., Haider, I., Taheri, R. e Conti, M. (2025). Federated learning under attack: Exposing vulnerabilities through data poisoning attacks in computer networks. *IEEE Trans. on Net. and Service Management*, 22(1):822–831.

[Pandya et al., 2023] Pandya, S. et al. (2023). Federated learning for smart cities: a comprehensive survey. *Sustainable Energy Technologies and Assessments*, 55:102987.

[Pang et al., 2025] Pang, X., Zhao, C., Wang, Z., Hu, J., Wang, Y., Wang, L., Wei, T., Ren, K. e Chen, C. (2025). poisafl: scalable poisoning attack framework to byzantine-resilient semi-asynchronous federated learning. Em *USENIX Security*, p. 6461–6479.

[Pillutla et al., 2022] Pillutla, K., Kakade, S. M. e Harchaoui, Z. (2022). Robust aggregation for federated learning. *IEEE Transactions on Signal Processing*, 70:1142–1154.

[Rieke et al., 2020] Rieke, N., Hancox, J., Li, W., Milletari, F., Roth, H. R., Albarqouni, S., Bakas, S., Galtier, M. N., Landman, B. A., Maier-Hein, K. et al. (2020). The future of digital health with federated learning. *NPJ digital medicine*, 3(1):119.

[Sattler et al., 2020] Sattler, F., Müller, K.-R., Wiegand, T. e Samek, W. (2020). On the byzantine robustness of clustered federated learning. Em *IEEE ICASSP 2020*, p. 8861–8865. IEEE.

[Shejwalkar e Houmansadr, 2021] Shejwalkar, V. e Houmansadr, A. (2021). Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. Em *Ndss*.

[Shen et al., 2016] Shen, S., Tople, S. e Saxena, P. (2016). Auror: Defending against poisoning attacks in collaborative deep learning systems. Em *Proceedings of the 32nd ACSAC '16*, p. 508–519, New York, NY, USA. ACM.

[Shi et al., 2022] Shi, J., Wan, W., Hu, S., Lu, J. e Zhang, L. Y. (2022). Challenges and approaches for mitigating byzantine attacks in federated learning. Em *2022 IEEE International Conference TrustCom*, p. 139–146. IEEE.

[Shokri et al., 2017] Shokri, R., Stronati, M., Song, C. e Shmatikov, V. (2017). Membership inference attacks against machine learning models. Em *2017 IEEE symposium on security and privacy (SP)*, p. 3–18. IEEE.

[Sun et al., 2021] Sun, L., Qian, J. e Chen, X. (2021). Ldp-fl: Practical private aggregation in federated learning with local differential privacy. Em *Proceedings of the Thirtieth IJCAI-21*, p. 1571–1578. Inter. Joint Conf. on Artificial Intell. Organization.

[Tran et al., 2018] Tran, B., Li, J. e Madry, A. (2018). Spectral signatures in backdoor attacks. *Advances in neural information processing systems*, 31.

[Truex et al., 2019] Truex, S., Baracaldo, N., Anwar, A., Steinke, T., Ludwig, H., Zhang, R. e Zhou, Y. (2019). A hybrid approach to privacy-preserving federated learning. Em *Proceedings of the 12th ACM workshop on artificial intelligence and security*, p. 1–11.

[Truex et al., 2020] Truex, S., Liu, L., Chow, K.-H., Gursoy, M. E. e Wei, W. (2020). ldp-fed: Federated Learning with Local Differential Privacy. Em *Proceedings of the third ACM inter. workshop on edge systems, analytics and networking*, p. 61–66.

[Wang et al., 2020] Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., Sohn, J.-y., Lee, K. e Papailiopoulos, D. (2020). Attack of the tails: Yes, you really can backdoor federated learning. Em *Advances in Neural Information Processing Systems*, volume 33, p. 16070–16084. Curran Associates, Inc.

[Wang et al., 2019] Wang, N., Xiao, X., Yang, Y., Zhao, J., Hui, S. C., Shin, H., Shin, J. e Yu, G. (2019). Collecting and analyzing multidimensional data with local differential privacy. Em *2019 IEEE 35th ICDE*, p. 638–649. IEEE.

[Wang et al., 2025] Wang, T.-H., Chen, P.-N. e Huang, Y.-C. (2025). Probabilistic byzantine attack on federated learning. *IEEE Transactions on Signal Processing*.

[Wang et al., 2022] Wang, Z., Huang, Y., Song, M., Wu, L., Xue, F. e Ren, K. (2022). Poisoning-assisted property inference attack against federated learning. *IEEE Transactions on Dependable and Secure Computing*, 20(4):3328–3340.

[Wen et al., 2023] Wen, J., Zhang, Z., Lan, Y., Cui, Z., Cai, J. e Zhang, W. (2023). A survey on federated learning: Challenges and applications. *International journal of machine learning and cybernetics*, 14(2):513–535.

[Wu et al., 2024] Wu, D., Bai, J., Song, Y., Chen, J., Zhou, W., Xiang, Y. e Sajjanhar, A. (2024). fedinverse: evaluating privacy leakage in federated learning. Em *The twelfth international conference on learning representations*.

[Xia et al., 2024] Xia, F., Liu, Y., Jin, B., Yu, Z., Cai, X., Li, H., Zha, Z., Hou, D. e Peng, K. (2024). Leveraging multiple adversarial perturbation distances for enhanced membership inference attack in federated learning. *Symmetry*, 16(12):1677.

[Xie et al., 2020] Xie, C., Koyejo, S. e Gupta, I. (2020). Zeno++: Robust fully asynchronous sgd. Em *Inter. Conference on Machine Learning*, p. 10495–10503. PMLR.

[Yaldiz et al., 2023] Yaldiz, D. N., Zhang, T. e Avestimehr, S. (2023). Secure federated learning against model poisoning attacks via client filtering. *arXiv preprint arXiv:2304.00160*.

[Yang et al., 2025] Yang, L., Miao, Y., Liu, Z., Liu, Z., Li, X., Kuang, D., Li, H. e Deng, R. H. (2025). Enhanced model poisoning attack and multi-strategy defense in federated learning. *IEEE Transactions on Information Forensics and Security*.

[Yang et al., 2019] Yang, Q., Liu, Y., Chen, T. e Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Trans. on Intell. Sys. and Tech.*, 10(2):1–19.

[Ye et al., 2023] Ye, M. et al. (2023). Heterogeneous federated learning: State-of-the-art and research challenges. *ACM Computing Surveys*, 56(3):1–44.

[Yin et al., 2018] Yin, D., Chen, Y., Kannan, R. e Bartlett, P. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. Em *International conference on machine learning*, p. 5650–5659. Pmlr.

[Zhang et al., 2019] Zhang, J., Chen, J., Wu, D., Chen, B. e Yu, S. (2019). Poisoning attack in federated learning using generative adversarial nets. Em *2019 18th IEEE TrustCom/BigDataSE*, p. 374–380.

[Zhang et al., 2025] Zhang, K., Xu, P. e Tian, Z. (2025). Distributional and byzantine robust decentralized federated learning. Em *2025 59th Annual CISS*, p. 1–6. IEEE.

[Zhao et al., 2020a] Zhao, B., Mopuri, K. R. e Bilen, H. (2020a). idlg: improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*.

[Zhao et al., 2025] Zhao, J. et al. (2025). The federation strikes back: a survey of federated learning privacy attacks, defenses, applications, and policy landscape. *ACM Computing Surveys*, 57(9):1–37.

[Zhao et al., 2020b] Zhao, P., Chen, P.-Y., Das, P., Ramamurthy, K. N. e Lin, X. (2020b). Bridging mode connectivity in loss landscapes and adversarial robustness. *arXiv preprint arXiv:2005.00060*.

[Zhao et al., 2020c] Zhao, Y. et al. (2020c). Local differential privacy-based federated learning for internet of things. *IEEE IoT Journal*, 8(11):8836–8853.

[Zhu et al., 2025] Zhu, G. et al. (2025). fedmia: an effective membership inference attack exploiting "all for one" principle in federated learning. Em *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 20643–20653.

[Zhu et al., 2019] Zhu, L., Liu, Z. e Han, S. (2019). Deep leakage from gradients. *Advances in neural information processing systems*, 32.