UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

THIAGO DORNELAS BORBA
Orientador: Dr. Pedro Henrique Lopes Silva
Coorientador: Dr. Joubert de Castro Lima

ESTUDO SOBRE *FEDERATED LEARNING*: ANÁLISE DA LITERATURA E REALIZAÇÃO DE TESTES UTILIZANDO O FLOWER

Ouro Preto, MG
2022

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

THIAGO DORNELAS BORBA

ESTUDO SOBRE *FEDERATED LEARNING*: ANÁLISE DA LITERATURA E REALIZAÇÃO DE TESTES UTILIZANDO O FLOWER

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Dr. Pedro Henrique Lopes Silva
Coorientador: Dr. Joubert de Castro Lima

Ouro Preto, MG
2022

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Thiago Dornelas Borba

Estudo sobre Federated Learning: Análise da literatura e realização de testes utilizando o Flower

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 24 de Março de 2023.

Membros da banca

Pedro Henrique Lopes Silva (Orientador) - Doutor - Universidade Federal de Ouro Preto
Joubert de Castro Lima (Coorientador) - Doutor - Universidade Federal de Ouro Preto
Reinaldo Silva Fortes (Examinador) - Doutor - Universidade Federal de Ouro Preto
Rodrigo César Pedrosa Silva (Examinador) - Doutor - Universidade Federal de Ouro Preto

Pedro Henrique Lopes Silva, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 24/03/2023.

Documento assinado eletronicamente por **Pedro Henrique Lopes Silva**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 27/03/2023, às 22:34, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0493050** e o código CRC **64852551**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Dedico este trabalho à minha mãe, Angélica, por ter me educado muito bem e incentivado meus estudos como ninguém, essa conquista também é sua!

Agradecimentos

Este trabalho representa um marco importante da minha graduação, gostaria de agradecer especialmente aos meus orientadores, Pedro Silva e Joubert Lima, por toda a paciência, suporte e dedicação para orientar esta pesquisa, sem dúvidas a ajuda de vocês foi fundamental para o meu aprendizado. Agradeço também ao Departamento de Ciência da Computação (DECOM) e a todos os professores que me orientaram ao longo dos últimos anos, foi um privilégio aprender com vocês.

"Imagine all the people sharing all the world..."

- John Lenon

Resumo

Federated Learning (FL) é uma abordagem para diferentes dispositivos treinarem modelos compartilhados de forma colaborativa, sem enviar seus dados para o servidor, apenas rodando um treinamento local no próprio dispositivo, dissociando assim o aprendizado de máquina da necessidade de armazenamento desses dados em larga escala na nuvem. A princípio, o presente trabalho apresenta uma introdução sobre FL, contextualizando seu surgimento. A fim de estabelecer um embasamento teórico, vários tópicos relacionados ao assunto são discutidos, incluindo uma comparação entre *Machine Learning* (ML), *Distributed Machine Learning* e FL. Além disso, é feita uma descrição sobre a arquitetura do Flower, uma plataforma de código aberto criado para facilitar o desenvolvimento de soluções baseadas em FL. Foram conduzidos experimentos com essa ferramenta utilizando duas bases de dados, ambas contendo imagens médicas de radiografia de tórax, uma voltada para reconhecimento de COVID-19 e outra para pneumonia. Os resultados desses experimentos foram comparados com os resultados obtidos através de testes utilizando ML centralizado sob as mesmas configurações. Embora os valores de acurácia e perda tenham sido similares, observou-se que o teste FL apresentou um tempo de execução maior, o que evidenciou os custos de comunicação entre cliente e servidor. Todos os experimentos realizados foram cuidadosamente documentados, incluindo as ferramentas e tecnologias utilizadas durante o processo. A principal contribuição dessa pesquisa reside na elaboração de conteúdo introdutório sobre FL, cobrindo tanto os aspectos teóricos quanto práticos da abordagem, incluindo os procedimentos necessários para utilizar o Flower e outras atividades inerentes à ferramenta escolhida.

Palavras-chave: Federated Learning. Machine Learning. Distributed Machine Learning. Flower.

Abstract

Federated Learning (FL) is an approach where different devices collaboratively train shared models without sending their data to the server, only running local training on the device itself, thus dissociating machine learning from the need for large-scale storage of data in the cloud. This paper provides an introduction to Federated Learning, contextualizing its emergence. To establish a theoretical foundation, various topics related to the subject are discussed, including a comparison between Machine Learning (ML), Distributed Machine Learning and FL. In addition, the article provides a description of the Flower architecture, an open-source platform designed to facilitate the development of FL solutions. Experiments were conducted using this tool on two databases containing medical chest X-ray images, one for COVID-19 recognition and the other for pneumonia. The results of these experiments were compared to those obtained through centralized ML tests under the same settings. Although the accuracy and loss values were similar, it was observed that the FL test presented a longer execution time, highlighting the costs of communication between the client and server. All experiments were carefully documented, including the tools and technologies used during the process. The main contribution of this research lies in the development of introductory content on FL, covering both theoretical and practical aspects of the approach, including the necessary procedures for using the Flower platform and other activities inherent to the chosen tool.

Keywords: Federated Learning. Machine Learning. Distributed Machine Learning. Flower.

Lista de Ilustrações

Lista de Tabelas

Lista de Códigos Fonte

Lista de Abreviaturas e Siglas

BlockFL	*Blockchained Federated Learning*
CNN	*Convolutional Neural Network*
DECOM	Departamento de Computação
DML	*Distributed Machine Learning*
FedAvg	*Federated Averaging*
FL	*Federated Learning*
IoT	*Internet of Things*
ML	*Machine Learning*
RPC	*Remote Procedure Call*
TFF	TensorFlow Federated
UFOP	Universidade Federal de Ouro Preto
VM	Máquina Virtual
IA	Inteligência Artificial
SAC	Serviço de Atendimento ao Cidadão
FC	API Federated Core
GCP	Google Cloud Platform

Sumário

1 Introdução

Cada vez mais, o uso de aprendizado de máquina, ou *Machine Learning* (ML), faz parte do nosso cotidiano. Essa ciência vem trazendo diversos avanços tecnológicos impactando diretamente em nossas vidas (JORDAN; MITCHELL, 2015). Alguns exemplos de aplicações de ML são:

- Recomendação de conteúdo: aplicativos de *streaming* fazendo um catálogo personalizado para você, de acordo com o que já assistiu (SUNNY et al., 2017).
- Detecção de fraudes: monitoramento de atividades suspeitas nas contas de clientes de um banco (THENNAKOON et al., 2019).
- Operações financeiras: identificação de transações financeiras lucrativas, como na bolsa de valores (SHEN; JIANG; ZHANG, 2012).
- Serviço de Atendimento ao Cidadão (SAC): robôs inteligentes utilizados para responder a problemas e dúvidas de clientes pelo chat (PEIXOTO, 2021).
- Veículos inteligentes: veículos que se locomovem de forma autônoma, sem um motorista (ZANTALIS et al., 2019).

Na abordagem tradicional de aprendizado de máquina, um modelo é treinado a partir de todos os dados disponíveis centralizados no servidor (MCMAHAN; RAMAGE, 2017). O que funciona bem quando o servidor possui todos esses dados disponíveis e consegue oferecer as predições em tempo hábil de acordo com o contexto.

O uso em massa de dispositivos móveis, como celulares e *tablets*, tem gerado uma enorme quantidade de dados de seus usuários (KIUKKONEN et al., 2010). Esses dados são de grande interesse para empresas, pois podem ser utilizados para treinamento de diversos modelos, uma vez que apresentam grande diversidade. Contudo, os dados produzidos frequentemente são sensíveis, isto é, informações que podem ser utilizadas para discriminar uma pessoa, além de serem gerados em larga escala, o que dificulta o seu processamento e armazenamento de forma centralizada, como acontece na abordagem tradicional de ML. Além disso, tendo em vista a necessidade de respostas rápidas na computação móvel, essa abordagem também não seria a ideal, uma vez que a comunicação entre o servidor e o dispositivo poderia ser lenta.

Identificando esse problema de ML com dados descentralizados, McMahan et al. (2016) introduz a abordagem *Federated Learning* (FL) para diferentes dispositivos treinarem modelos compartilhados de forma colaborativa, sem enviar seus dados para o servidor, apenas rodando

um treinamento local no próprio dispositivo, dissociando assim o aprendizado de máquina da necessidade de armazenamento desses dados em larga escala na nuvem.

No processo de treinamento FL, o modelo local é treinado em cada dispositivo, que envia uma atualização ao servidor após cada rotina de treinamento. Esse, por sua vez, faz a agregação de todas as atualizações recebidas, atualiza o modelo global, envia essa atualização aos dispositivos e assim por diante (KAIROUZ et al., 2021). Conforme mostrado na Figura 1.1, FL pode ser usado em diversos cenários, como por exemplo:

- Veículos inteligentes: os veículos são treinados em tempo real com informações sobre a estrada e o tráfego. Imagine uma rodovia onde todos os veículos são autônomos e compartilham informações entre si: um mundo sem acidentes (DU et al., 2020).
- Saúde: treinar um modelo global utilizando dados de vários lugares diferentes tornaria possível que as máquinas fizessem diagnósticos de casos médicos, ou até mesmo fazer predições, como no caso de relógios inteligentes que monitoram vários aspectos do nosso organismo, eles poderiam estar sendo treinados em tempo real para interpretar os sinais monitorados (RIEKE et al., 2020).
- Indústrias: predição de manutenção de equipamentos e difusão ainda maior da automação de processos, uma vez que seria possível treinar sistemas embarcados que são usados para diversas medições e tarefas (NGUYEN et al., 2021).

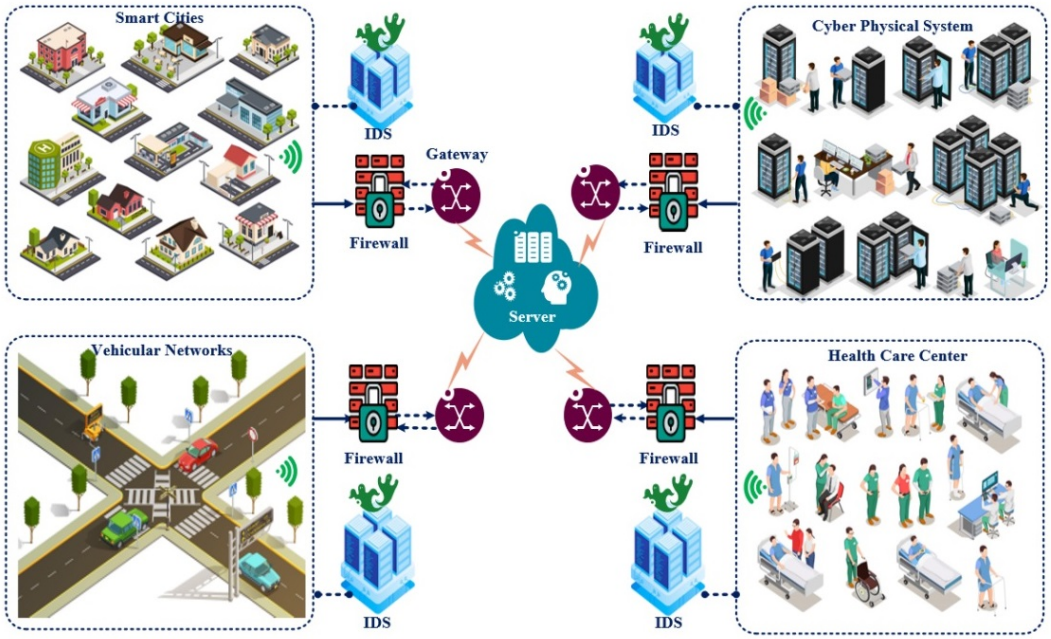


Figura 1.1 – *Federated Learning* aplicado em diversas áreas.
Fonte: (MATHUR et al., 2021).

O Flower (FLOWER, 2019) é uma plataforma de código aberto para ML criada com o intuito de facilitar a implementação de modelos e ambientes FL, facilitando as pesquisas e

simulações acerca do tema. Neste trabalho o Flower será utilizado para criar um ambiente onde seja possível treinar um modelo FL utilizando um servidor e diferentes clientes, respeitando a privacidade dos dados e as diretrizes de aprendizado descentralizado e distribuído. Foram realizados testes com diferentes bases de dados contendo imagens médicas a fim de validar o funcionamento do ambiente implementado e estabelecer comparações entre o treinamento com ML centralizado e o treinamento com FL. Além disso, também foi implementado um tutorial (Apêndice A) que permite a execução de um treinamento utilizando Flower de forma simples.

1.1 Justificativa

Uma das principais vantagens do *Federated Learning* pode ser vista como superar a necessidade de acesso direto aos dados no treinamento de um modelo, o que aumenta a segurança da privacidade dos usuários que estão fornecendo esses dados e também evita o problema de armazenamento gerado pela centralização de dados em larga escala.

Sendo um tema extremamente recente no âmbito da pesquisa (MCMAHAN et al., 2016), existem diversos desafios acerca da implementação de um ambiente para o treinamento de um modelo FL, que em muitas vezes não são decorrentes do aprendizado de máquina em si, mas de questões de infraestrutura e tratativa de dados, como o uso de dados sensíveis e/ou em larga escala, como citado no parágrafo anterior.

Dito isso, para a utilização dessa abordagem, não basta apenas o trabalho dos cientistas de dados, que implementam o código referente ao ML, mas também toda a criação e configuração da infraestrutura necessária para a viabilização de um treinamento FL, o que acaba sendo complicado para os cientistas, que não são especialistas nessa parte e muitas vezes não conseguem implantar suas aplicações.

Portanto, é evidente a importância de pesquisas e trabalhos em prol da solução de problemas de infraestrutura acerca de FL, tanto para ajudar os cientistas de dados fazerem seu trabalho sem a necessidade de se preocuparem com assuntos que não dominam, quanto para contribuir para a comunidade científica como um todo, visto que o material conhecido sobre o tema é bem escasso.

1.2 Objetivos

O objetivo principal deste trabalho é aplicar técnicas de FL a problemas de ML respeitando a privacidade dos dados, bem como descrever todos os conhecimentos adquiridos no decorrer do processo, juntamente com os experimentos realizados e as ferramentas utilizadas. Para isso, tem-se os seguintes objetivos específicos:

- Criar um tutorial (Apêndice A) simplificado para execução de um experimento FL utili-

zando Flower.

- Comparar a utilização de ML e FL por meio da realização de testes utilizando as duas abordagens.
- Avaliar uso de FL em dispositivos móveis.

1.3 Organização da Monografia

Essa monografia se organiza da seguinte forma:

Capítulo 2: Apresenta a contextualização sobre *ML*, *Distributed ML*, *FL*, Flower e por fim os trabalhos relacionados ao tema.

Capítulo 3: Detalha todos os processos realizados na parte prática desta pesquisa.

Capítulo 4: Apresenta detalhes dos experimentos e análise dos resultados.

Capítulo 5: Apresenta a conclusão acerca dos estudos e contribuições realizados, além de discutir possíveis trabalhos futuros.

2 Fundamentação Teórica

Com o objetivo de estabelecer uma base teórica sólida, este capítulo discute diversos tópicos relacionados a *Federated Learning*. A princípio, esses tópicos são apresentados de forma a contextualizar as tecnologias e os desafios que levaram ao desenvolvimento do FL. A Seção 2.1 apresenta os conceitos fundamentais de aprendizado de *Machine Learning*. Em seguida, a Seção 2.2 aborda o tema *Distributed Machine Learning* (DML), relacionando-o aos desafios enfrentados pelo ML. A Seção 2.3, por sua vez, estabelece uma comparação entre o FL e os tópicos discutidos anteriormente, aprofunda nos conceitos de FL e em seu funcionamento, além de apresentar o *framework* Flower e descrever um pouco de sua arquitetura. Nas Seções 2.4, 2.5 e 2.6 são descritas ferramentas que podem ser úteis para trabalhar com FL na prática. Por fim, a Seção 2.7 faz uma revisão da literatura, discutindo sobre obras que abordam o surgimento de FL, algoritmos que servem como base para seu funcionamento, aplicações em diferentes áreas e *frameworks* disponíveis para implementação do FL.

2.1 *Machine Learning*

A Inteligência Artificial (IA) opera sob a premissa de que todas as formas de aprendizado e inteligência podem ser definidas com precisão na medida em que uma máquina pode replicá-las (DICK, 2019). Assim, o objetivo primordial desta área é criar modelos computacionais com capacidade de reproduzir comportamentos inteligentes, como compreensão de texto (WANG; BABENKO; BELONGIE, 2011), reconhecimento de elementos visuais (GRILL-SPECTOR; KANWISHER, 2005) ou mesmo tomada de decisão (PHILLIPS-WREN, 2012).

Machine Learning é uma forma de criar IA e surge da teoria de que os computadores podem aprender à medida que processam dados, sem precisar ser programados para tarefas específicas (MITCHELL; MITCHELL, 1997). O processo de ML consiste em criar um modelo de reconhecimento de padrões e treiná-lo, fornecendo-lhe dados e um algoritmo capaz de ponderar e aprender com esses dados. Por exemplo, se desejar reconhecer imagens de órgãos afetados por uma doença, é possível treinar um modelo com imagens de órgãos que foram e não foram afetados pela doença e, em seguida, utilizar esse modelo para determinar se um órgão qualquer foi afetado ou não. Esse exemplo caracteriza uma forma de realizar ML, existem várias outras. Com base no artigo Silva (2021), algumas dessas formas são descritas a seguir:

- Aprendizado supervisionado: funciona como no exemplo citado acima, onde o objetivo é classificar dados de entrada, produzindo uma saída. É necessário fornecer ao sistema dados de treinamento e seus respectivos rótulos (classes) para que ele possa classificar

novos dados recebidos, como acontece em casos de classificação de *spam* e sistemas de diagnóstico médico.

- Aprendizado não supervisionado: O treinamento é feito a partir de dados não classificados (sem rótulos), e o sistema deve ser capaz de descobrir padrões para agrupá-los com base em características semelhantes. Lidar com dados não classificados frequentemente significa lidar com um conjunto grande de dados, tornando a complexidade computacional primordial.

- Aprendizado por reforço: o modelo é treinado para tomar uma sequência de decisões a fim de resolver um problema. É utilizado um sistema de recompensas e o computador aprende por meio da experiência, ou seja, tentativa e erro. A cada tentativa, é avaliado o progresso em relação ao estado anterior e ao estado desejado, determinando uma recompensa ou penalidade que orientará o computador a repetir ou evitar tais ações, visando receber a maior recompensa possível.

2.1.1 Rede Neural Convolucional

Um dos expoentes em ML é o *Deep Learning* (ALZUBAIDI et al., 2021), tendo as Redes Neurais Convolucionais como principal técnica para processamento de imagens considerando o aprendizado supervisionado. Dito isso, foi a técnica escolhida para esta pesquisa.

Redes Neurais Convolucionais, do inglês *Convolutional Neural Networks* (CNN), são modelos com arquitetura capaz de aprender diretamente dos dados, dispensando a necessidade de extração manual de atributos (ALBAWI; MOHAMMED; AL-ZAWI, 2017). Cada camada da CNN é responsável por extrair determinadas informações da entrada, passando de camada a camada, onde a saída de uma camada é a entrada da camada subsequente.

De acordo com Lang (2021), as CNNs são amplamente utilizadas no processamento de imagens, principalmente por conta de suas camadas convolucionais que reduzem significativamente o número de parâmetros a serem considerados, facilitando a identificação de padrões nas imagens. Além disso, mesmo com a redução das dimensões dos dados, as principais características da imagem são preservadas.

Conforme exemplificado na Figura 2.1, a camada convolucional consiste na aplicação de diversos filtros na imagem. Esses filtros são máscaras que se deslocam ao longo da imagem, como uma janela deslizante, processando cada sub-imagem e multiplicando seus valores pelo filtro definido. A saída dessa camada é uma imagem de tamanho menor ou igual ao da imagem de entrada.

Feito a convolução, normalmente o próximo passo é a camada de *Pooling*. Ela é muito parecida com a camada convolucional (YAMASHITA et al., 2018), a grande diferença é que ela processa o valor médio ou máximo de cada sub-imagem, dependendo da configuração escolhida,

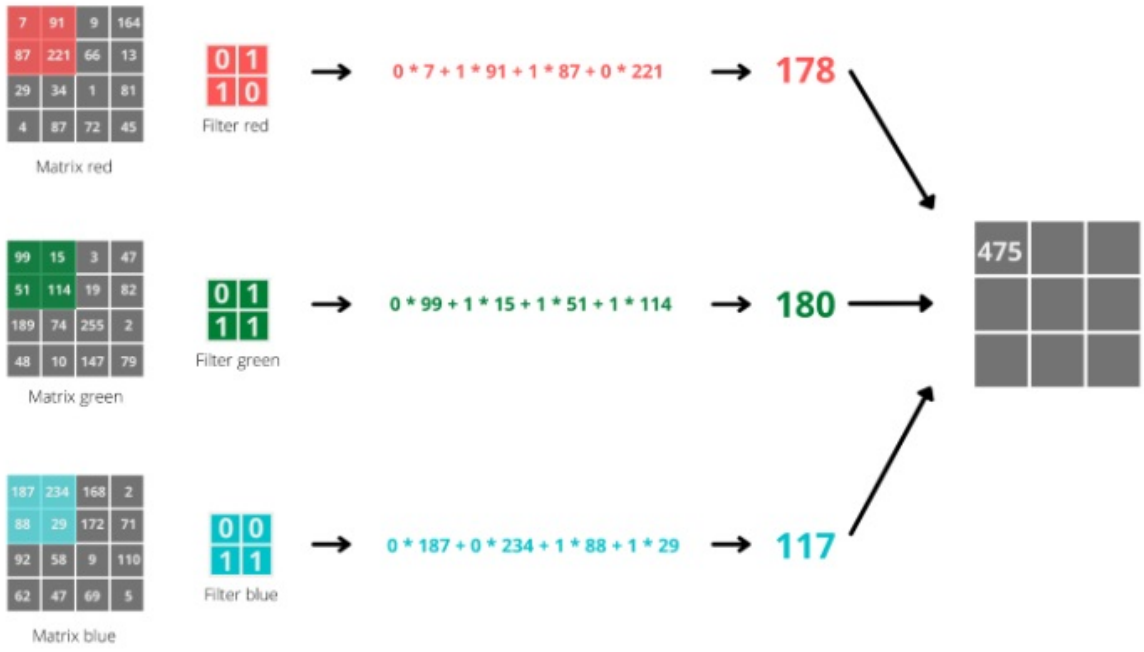


Figura 2.1 – Convolução de uma imagem 4x4x3 utilizando filtros 2x2.
Fonte: (LANG, 2021).

o que preserva algumas características da imagem que podem ser importantes para resolver o problema.

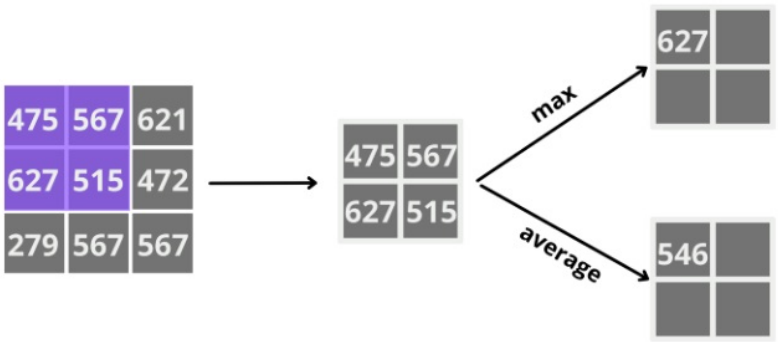


Figura 2.2 – Dois exemplos de aplicação do *Pooling* em uma matriz 3x3, um pegando o maior valor e outro pegando a média, ambos utilizando filtros 2x2,
Fonte: (LANG, 2021).

Após reduzir o número de parâmetros com a camada de *Pooling*, pode-se aplicar uma camada de *Flattening* para converter o resultado em um vetor unidimensional. Esse vetor unidimensional é então utilizado na camada densa, que é uma camada totalmente conectada da rede neural, utilizando uma função de ativação para classificar a imagem de acordo com o número de classes desejadas. A função de ativação escolhida dependerá do número de classes, como a

Sigmóide usada para trabalhar com duas classes e a *Softmax* que distribui as probabilidades para múltiplas classes (CECCON, 2020).

2.1.2 Desafios de ML

Além dos tipos de aprendizado previamente descritos, existem também diversas abordagens arquiteturais utilizadas em ML. Na abordagem tradicional, ilustrada na Figura 2.3, todo o processo de treinamento ocorre de forma centralizada em um único computador, que detém todos os dados disponíveis, bem como o código de treinamento, o modelo e o *hardware* necessário para realizar o processo. Com o passar do tempo e a evolução rápida da computação móvel e em rede, a quantidade de dados produzidos e transportados tornou-se significativamente maior, originando o termo *Big Data* (ORACLE, 2022). Tendo em vista a necessidade de se ter tais dados disponíveis para treinamento no servidor e considerando que eles geralmente precisam ser coletados em tempo real e de forma contínua, além de serem privados em muitos casos, é evidente que a abordagem centralizada apresenta algumas limitações em termos de escalabilidade, eficiência, latência e violação da privacidade dos dados.

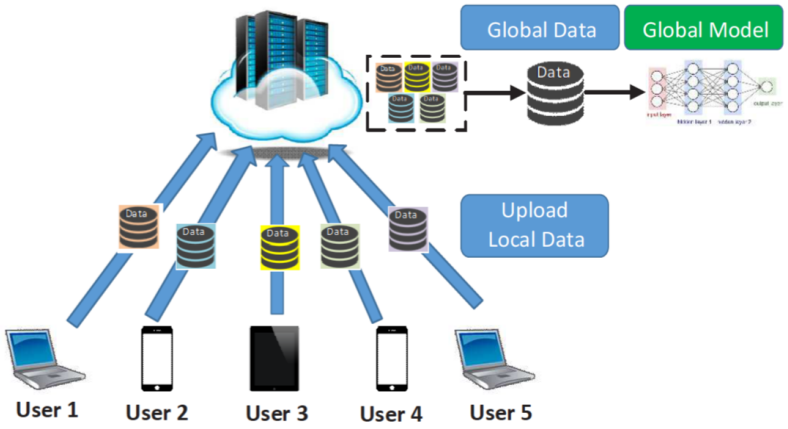


Figura 2.3 – Arquitetura do ML centralizado, onde vários clientes enviam seus dados para o servidor realizar o treinamento.
Fonte: (DIGEST, 2021).

2.2 *Distributed Machine Learning*

Distributed Machine Learning (DML) é um subcampo de ML que envolve o uso de múltiplos dispositivos ou nós de computação para treinar um modelo colaborativamente. Em DML, os dados são distribuídos em várias máquinas ou nós e o processo de aprendizagem é descentralizado, com cada nó processando um subconjunto dos dados (VERBRAEKEN et al., 2020).

A utilização de DML surge em cenários onde a quantidade de dados é grande demais para ser processada em uma única máquina ou quando os recursos computacionais necessários

para treinar um modelo são muito elevados. Nesses casos, a distribuição dos dados e do processo de aprendizagem em várias máquinas pode acelerar significativamente o processo de treinamento e melhorar a precisão do modelo resultante.

Os algoritmos DML podem ser categorizados em dois tipos principais: paralelos e distribuídos. Nos algoritmos paralelos, várias cópias do mesmo modelo são treinadas simultaneamente em diferentes subconjuntos dos dados, e os modelos resultantes são combinados para formar um modelo final. Nos algoritmos distribuídos, os dados são particionados em várias máquinas, e cada máquina treina um modelo separado em seu subconjunto dos dados. Posteriormente, os modelos resultantes são então combinados usando técnicas como *model averaging*.

Alguns *frameworks* como Apache Hadoop (FOUNDATION, 2006-2023a), Apache Spark (FOUNDATION, 2006-2023b) e TensorFlow (GOOGLE, 2020) fornecem ferramentas para gerenciar a computação distribuída, lidar com a distribuição e comunicação de dados e otimizar o desempenho do processo de aprendizagem. Portanto, essas estruturas podem ser muito úteis no contexto de DML.

De maneira geral, DML é um campo em rápido crescimento que está transformando a forma como os modelos de aprendizagem de máquina são treinados e implantados. No entanto, também apresenta seu próprio conjunto de desafios, como a consistência dos dados, sobrecarga de comunicação e balanceamento de carga.

Assim sendo, DML apresenta diversas vantagens em relação à abordagem tradicional de ML, incluindo tempos de treinamento mais rápidos, maior escalabilidade e maior tolerância a falhas. No entanto, não resolve a questão da violação da privacidade dos dados, um dos problemas destacados anteriormente.

2.3 *Federated Learning*

Como mencionado anteriormente, o *Federated Learning* surge como uma alternativa para dispositivos treinarem um modelo sem a necessidade de enviar seus dados para o servidor. Conforme mostrado na Figura 2.4, funciona da seguinte forma:

- Servidor seleciona clientes elegíveis para o treinamento.
- Clientes recebem o modelo atualizado do servidor e fazem o treinamento usando dados locais.
- Servidor recebe as atualizações dos clientes, faz a agregação delas utilizando estratégias FL.
- Clientes recebem o modelo agregado pelo servidor.

Pode-se dizer que FL faz um processo inverso em relação à abordagem tradicional de ML - ao invés de coletar dados dos clientes para treinar um modelo em um único computador, propõe-se o envio do modelo para esses clientes treinarem localmente utilizando seus dados e orquestra-se a comunicação deles com o servidor (MCMAHAN et al., 2016). FL tem algumas semelhanças com o processo de DML explicado na Seção 2.2, pode ser considerado um tipo de aprendizado de máquina distribuído. No entanto, enquanto o principal objetivo de DML é acelerar o processo de treinamento dividindo a carga de trabalho entre diferentes máquinas e permitindo uma maior escalabilidade, FL foca em garantir a não violação da privacidade dos dados.

O surgimento do termo aborda muitos problemas já existentes e introduz novos, que são interdisciplinares, uma vez que envolvem não apenas ML, mas também otimização, criptografia, segurança, privacidade, comunicação, entre outros. Portanto, é um tema que envolve, sobretudo, a colaboração de diversas áreas do conhecimento.

Segundo Kairouz et al. (2019), existem duas características principais que definem FL: (i) os dados são gerados localmente e permanecem descentralizados; e (ii) possui um servidor que coordena e organiza o treinamento. Em seguida descreveremos as duas principais variantes definidas até o momento, que mantém essas duas características, mas diferem em outras.

2.3.1 *Cross-device*

Nessa variante a escala de clientes é muito grande, podendo chegar até 10^{10} de acordo com Kairouz et al. (2019). Os clientes são dispositivos móveis ou dispositivos IoT e normalmente

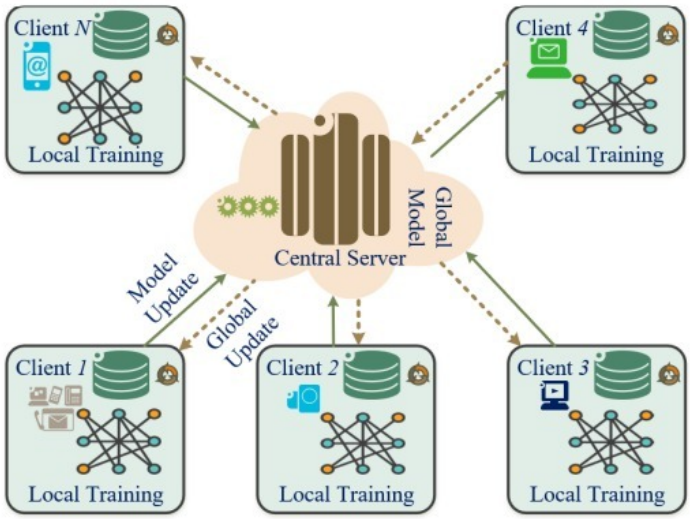


Figura 2.4 – Arquitetura FL, onde vários clientes treinam um modelo localmente com seus próprios dados e enviam as atualizações para o servidor, que atualiza o modelo global e o envia para os clientes.
Fonte: (AGRAWAL et al., 2021).

possuem recursos limitados, como bateria e conexão à internet. Nesse caso é necessário ter uma grande preocupação com a experiência do usuário. Para não serem impactados, os dispositivos devem estar carregando e conectados a uma rede com banda larga. Apenas uma fração dos clientes estará disponível a cada rodada de treinamento e é esperado que uma porcentagem desses clientes perca a conexão ou apresente outro tipo de erro durante o processo. Os clientes não podem ser indexados diretamente, uma vez que estão em redes distintas das quais o servidor não tem conhecimento. Por fim, o maior desafio e gargalo dessa variante acaba sendo a comunicação, sendo esse tema portanto um grande alvo de pesquisas que tangem ao assunto.

2.3.2 *Cross-silo*

Caracteriza um cenário onde uma ou mais empresas querem treinar um modelo de forma colaborativa, mas não podem compartilhar seus dados diretamente. Normalmente o número de clientes não é muito grande, porém é ainda mais sensível acerca de segurança e privacidade, uma vez que esses dados podem ser extremamente valiosos e um vazamento acarretaria num prejuízo muito grande. Outra grande questão em relação a essa variante é a política de recompensa diante da quantidade de dados que cada cliente contribui para o treinamento, visto que não seria atrativo para empresas que oferecem uma maior quantidade de dados ganharem o mesmo que empresas que oferecem menos (KAIROUZ et al., 2019).

Podemos concluir que ambas as variantes seguem a mesma arquitetura cliente-servidor, a grande diferença são os tipos de clientes que cada uma delas esperam e suas necessidades específicas, o que levanta problemas diferentes, como a dinâmica de comunicação em *cross-device* e a política de recompensas em *cross-silo*.

2.3.3 Flower

Com o Flower, o processo FL pode ser descrito como uma alternância entre computações globais e locais (BEUTEL et al., 2022). As computações globais são executadas no lado do servidor e responsáveis por coordenar o treinamento utilizando os clientes disponíveis. Computações locais são processos comuns de ML executados em cada cliente para treinar o modelo com seus próprios dados.

A Figura 2.5 mostra uma infraestrutura implementada para executar treinamentos em um ambiente com clientes heterogêneos. No lado do servidor nota-se três componentes:

- *Strategy*: se trata do algoritmo FL utilizado para agregar os resultados recebidos dos clientes. É possível escolher um algoritmo já oferecido pelo *framework* (como *FedAvg*) ou implementar um próprio.
- *Loop FL*: é quem orquestra tudo, embora não tome nenhuma decisão de como proceder. Ele é responsável por solicitar a configuração de cada rodada de treinamento à camada

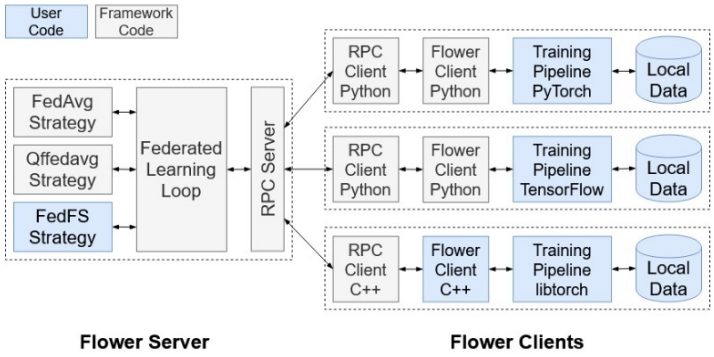


Figura 2.5 – Exemplo de como a arquitetura do Flower pode ser utilizada, nesse caso com três clientes implementados com linguagens e *frameworks* diferentes.
Fonte: (MATHUR et al., 2021).

Strategy, enviar aos clientes selecionados, receber as atualizações e delegar a agregação para a *Strategy*, que por sua vez, atualizará o modelo global.

- Servidor *Remote Procedure Call* (RPC): RPC define um protocolo para execução remota de procedimentos em computadores conectados por uma rede, é o meio utilizado pelo Flower para realizar a comunicação entre servidor e cliente.

Conectando-se a esse servidor RPC nota-se três clientes heterogêneos: (i) implementado em Python com *PyTorch*; (ii) implementado em Python com *TensorFlow*; (iii) implementado em C++ com *libtorch*. Esses clientes apenas esperam por instruções do servidor e executam as funções de treinamento e avaliação definidas localmente, ainda que seja possível também que o servidor as forneça.

Outro componente muito importante no lado do servidor, mostrado na Figura 2.6, é o *ClientManager*. Ele armazena um conjunto de objetos *ClientProxy*, que representam cada cliente conectado, e é por meio dele que o servidor consegue escolher os clientes para cada rodada de treinamento.

Flower foi projetado para ser um *framework* customizável e flexível, de forma a abstrair vários componentes para que eles possam ser substituídos ou aprimorados da forma mais conveniente para o desenvolvedor (BEUTEL et al., 2022). Dito isso, as principais características da ferramenta são:

- ML *framework-agnostic*: permite a utilização de diversos *frameworks* para ML, facilitando inclusive a migração de trabalhos existentes de ML para FL.
- *Client-agnostic*: permite que clientes se conectem utilizando diferentes linguagens de programação, sistemas operacionais ou *hardware*, de forma que a integração seja feita apenas por meio de protocolo, independente de outros fatores.

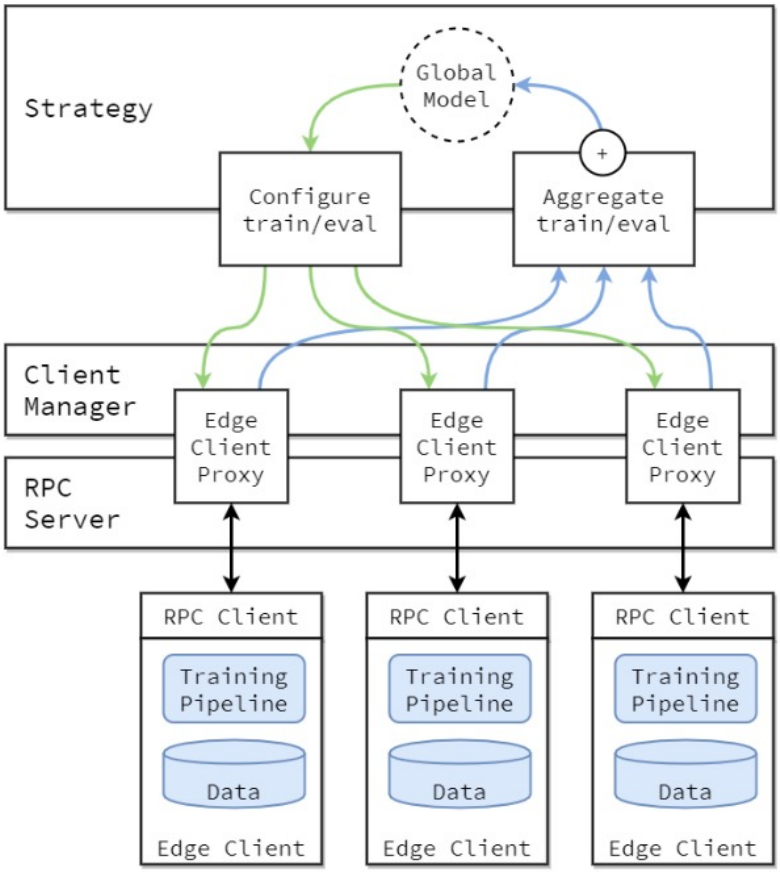


Figura 2.6 – Arquitetura Flower evidenciando os processos que ocorrem no lado do servidor.
Fonte: (FLOWER, 2019).

- *Privacy-agnostic*: permite a utilização de abordagens diferentes para garantir a privacidade dos dados.
- *Serialization-agnostic*: Os clientes recebem as mensagens como um vetor de *bytes*, que pode ser abstraído. Dessa forma, a troca de mensagens ocorre independente da linguagem de programação e também oferece ao usuário a possibilidade de implementar uma nova forma de serialização.
- Flexibilidade na comunicação: Ainda que a implementação atual utilize *gRPC* (*framework* RPC criado pelo Google (RYAN, 2015)) para comunicação, a arquitetura do Flower permite que um cliente se comunique utilizando um protocolo RPC próprio. A grande chave para que isso seja possível é a utilização da abstração chamada *ClientProxy*. Cada implementação dessa abstração guardará detalhes de como é feita a comunicação com o respectivo cliente. Dessa forma, o servidor consegue operar simultaneamente com clientes conectados via *gRPC* e clientes conectados por outro protocolo RPC.

2.4 Máquinas Virtuais

As máquinas virtuais (VMs) são um recurso importante em computação em nuvem, que permitem a execução de um sistema operacional ou aplicativo em um ambiente virtualizado. Elas podem ser criadas e gerenciadas em provedores de serviços em nuvem, permitindo que usuários acessem recursos de computação de alto desempenho sem precisar comprar e manter sua própria infraestrutura.

Existem vários provedores de serviços em nuvem que oferecem períodos de teste gratuitos para utilização seus serviços, incluindo, por exemplo, máquinas virtuais, armazenamento, banco de dados, análise de dados e serviços de inteligência artificial. Abaixo, são listados alguns exemplos:

- *Amazon Web Services* - oferece um período de teste gratuito de 12 meses.
- *Microsoft Azure* - oferece um período de teste gratuito de 30 dias.
- *Google Cloud Platform* (GCP) - oferece um período de teste gratuito de 90 dias.
- *IBM Cloud* - oferece um período de teste gratuito de 30 dias.
- *Oracle Cloud* - oferece um período de teste gratuito de 30 dias.
- *Alibaba Cloud* - oferece um período de teste gratuito de 30 dias.

2.5 Notebook Python

Os *notebooks* Python são criados a partir de células, que podem conter código Python executável, texto formatado em *Markdown* ou HTML, equações matemáticas em LaTeX, visualizações interativas, imagens, vídeos e outros tipos de conteúdo. As células podem ser executadas individualmente ou em conjunto, permitindo a exploração interativa de dados e a criação de relatórios interativos. É uma ferramenta muito popular principalmente entre cientistas de dados, pesquisadores acadêmicos e outros profissionais que trabalham com análise de dados no geral, visto que oferece um conjunto de facilidades para executar código, documentar e ainda visualizar resultados e dados de forma personalizável.

O *notebook* mais comumente utilizado é o Jupyter Notebook, que suporta não apenas Python, mas também outras linguagens de programação, como R, Julia e várias outras. O Jupyter Notebook é uma ferramenta poderosa e flexível que permite a análise de dados, a criação de modelos de ML e a exploração de dados em geral. Ele também possui uma grande comunidade de usuários e desenvolvedores, o que significa que é personalizável e possui muitos recursos disponíveis.

Outra opção popular de *notebook* é o Google Colab, que é um ambiente de desenvolvimento baseado em nuvem que oferece acesso a recursos de hardware poderosos, como GPUs e TPUs, para acelerar a execução de modelos de ML e outras tarefas computacionais intensivas.

2.6 Ambiente Virtual Python

Um ambiente virtual Python é uma instância isolada do Python que permite que os usuários criem um ambiente de desenvolvimento específico para cada projeto. Ele é usado para isolar dependências de pacotes específicos do projeto, mantendo-os separados do sistema Python global e de outros projetos.

Isso é importante porque diferentes projetos podem ter dependências conflitantes ou podem exigir versões diferentes do mesmo pacote. Ter ambientes virtuais Python separados para cada projeto pode ajudar a evitar conflitos e garantir que o projeto tenha todas as dependências necessárias para funcionar corretamente. Nos primeiros passos ao desenvolver FL com Python, uma das maiores dificuldades foi a resolução desses conflitos, para evitar tais problemas, é recomendado o uso de alguma ferramenta que permita a criação desses ambientes virtuais.

Existem várias ferramentas de gerenciamento de ambiente virtual Python disponíveis, como *virtualenv*, *pyenv*, *pipenv* e o próprio *venv* do Python. Essas ferramentas permitem criar, ativar e gerenciar diferentes ambientes virtuais Python.

Por exemplo, para criar um ambiente virtual usando o *venv*, abra o terminal e navegue até o diretório do projeto. Em seguida, digite o seguinte comando:

```
python3 -m venv env
```

Este comando cria um novo ambiente virtual chamado *env* no diretório atual. Para ativar o ambiente, basta digitar:

```
source env/bin/activate
```

Isso mudará o *prompt* do terminal para indicar que você está em um ambiente virtual. Agora, todas as dependências instaladas pelo *pip* serão instaladas apenas neste ambiente virtual. Depois de terminar de trabalhar em um projeto e quiser desativar o ambiente virtual, você pode executar o seguinte comando:

```
deactivate
```

2.7 Trabalhos Relacionados

O artigo (MCMAHAN et al., 2016) apresenta uma estratégia para agregação de modelos de ML, logo, essa abordagem permite a agregação de gradientes de diferentes modelos para atualizar um modelo global. No contexto do artigo, o objetivo é treinar um modelo centralizado

usando um conjunto de dados distribuído em dispositivos, sem compartilhar explicitamente os dados brutos. O algoritmo proposto no artigo é chamado de *FederatedAveraging* (FedAvg), que usa uma média ponderada dos gradientes dos modelos locais dos clientes, em que o peso é o tamanho do conjunto de dados local de cada cliente. Essa abordagem tem a vantagem de manter a privacidade dos dados de cada dispositivo, uma vez que apenas os gradientes são compartilhados e não os dados brutos. McMahan et al. (2016) também apresenta resultados experimentais de treinamentos realizados em diversos contextos diferentes, mostrando que o método proposto pode alcançar resultados comparáveis aos métodos tradicionais de ML com uma carga de comunicação significativamente menor. Por fim, é importante ressaltar que McMahan et al. (2016) foi o artigo que introduziu o termo *Federated Learning*, portanto essa abordagem proposta proposta foi de suma importância para a evolução das pesquisas na área.

Li et al. (2020b) propõe uma nova abordagem denominada FedProx, que tem como objetivo melhorar a eficiência e escalabilidade da aprendizagem federada em ambientes de rede heterogêneos. Essa abordagem introduz um mecanismo de penalidade de proximidade para melhorar a convergência e o desempenho do modelo. A penalidade de proximidade é um termo de regularização adicionado ao processo de otimização do modelo para evitar que os parâmetros se desviem muito dos valores da iteração anterior. O FedProx utiliza um hiperparâmetro denominado *proximal penalty strength* para controlar a força da penalidade de proximidade. O artigo (LI et al., 2020b) demonstra experimentalmente que o FedProx supera o FedAvg em cenários de aprendizagem federada com clientes heterogêneos, onde as capacidades e os dados dos clientes podem ser diferentes. O FedProx pode ser facilmente implementado como uma extensão do FedAvg, pois utiliza a mesma abordagem de atualização de modelo baseada em média ponderada dos gradientes locais dos clientes.

FL tem sido aplicado em diversas áreas, desde ML tradicional até na Indústria 4.0. Li et al. (2020a) aponta o crescimento de FL dado a falta de abordagens distribuídas de ML que resolvem o problema de quebra de privacidade dos dados. Também mostra o quanto esse crescimento agrega nas iniciativas de *Internet of things* (IoT) e na Indústria 4.0, uma vez que FL resolve um grande problema do ML nessas áreas que é a centralização dos dados, visto que em grande parte das vezes estão espalhados entre vários dispositivos, como sistemas embarcados. Por outro lado, também aponta que, embora várias plataformas e ferramentas FL estão sendo desenvolvidas, as pesquisas a cerca do tema são feitas principalmente por cientistas de dados, que focam mais na parte de ML do que na aplicação do FL em si, dos desafios por trás da engenharia por exemplo, o que acaba atrasando a aplicação em ambientes reais, como das indústrias.

Ainda na mesma linha de pensamento de Li et al. (2020a), Aledhari et al. (2020) levanta que mesmo com o número de pesquisas aumentando significativamente, ainda não tivemos o progresso suficiente para entender FL mais a fundo, uma vez que o assunto é muito recente e existem poucas aplicações reais. Assim, não conseguimos ter uma visão geral do assunto e entender exatamente como pode beneficiar as indústrias. Dito isso, o artigo (ALEDHARI et al.,

2020) tenta contribuir para a uma compreensão maior acerca da aplicabilidade e utilidade de FL. Alguns casos de uso mencionados são:

- Reconhecimento de imagens médicas: nesse exemplo é descrito um caso de uso para detecção de doenças cardíacas reumáticas no qual é utilizada a plataforma *ATMOSPHERE*, que fornece uma interface simplificada para treinamento e construção dos modelos.
- Detecção de anomalias para IoT: esse caso de uso procura resolver um problema de vulnerabilidade de dispositivos IoT detectando atividades maliciosas, um dos grandes desafios nesse caso foi lidar com dados heterogêneos, visto que é necessário reconhecer as particularidades de cada dispositivo.

Por fim, é observado que cada caso de uso tem suas próprias restrições e termos de software e hardware, o que pode resultar em soluções bem diferentes e impactar diretamente na performance.

Grande parte das abordagens de FL propostas, mesmo superando a necessidade de centralizar os dados, ainda possuem a dependência de um servidor central para orquestrar o treinamento e atualizar o modelo global. Em (KIM et al., 2019) é introduzida uma nova arquitetura chamada *blockchained* FL (BlockFL), que propõe a troca de atualizações dos modelos locais entre os dispositivos por meio da rede *blockchain*. Dessa forma, além de descartar a necessidade de um servidor, também apresenta uma política de compensação, de modo que cada dispositivo receba de acordo com o que está oferecendo. Outra vantagem dessa arquitetura está na validação dos resultados de treinamento local de cada dispositivo, ou seja, para autenticar o contrato de troca só é necessário a validação dos dados oferecidos.

Chen, Chuang e Wu (2020) descrevem alguns problemas existentes para aplicar ML utilizando dados de dispositivos IoT, como a privacidade dos dados, latência alta, segurança da informação, entre outros. Em seguida, aponta que FL será importante para o desenvolvimento de cidades inteligentes, uma vez que possibilita uma interação colaborativa entre diversos tipos de dispositivos sem precisar de acesso aos dados de cada um. Dessa forma seria possível melhorar diversos aspectos dentro do conceito de cidade inteligente, por exemplo, melhorar os sistemas de transporte e comunicação, ajudar no combate a fraudes no sistema financeiro, intensificar desenvolvimento de tecnologias de saúde inteligente, entre outros. Por fim, como próximo passo para viabilizar FL no contexto de cidade inteligente descrito, Chen, Chuang e Wu (2020) pontuam a necessidade de avançar em alguns campos como segurança, eficiência do algoritmo e distinção dos dados (para desenvolver diferentes políticas de compensação).

Para facilitar a implementação de FL, alguns *frameworks* foram propostos, tais como o Flower (BEUTEL et al., 2022) e TensorFlow Federated (GOOGLE, 2020).

Em (BEUTEL et al., 2022), é apontada uma grande disparidade entre ambientes FL de pesquisa e produção, uma vez que poucos trabalhos de pesquisa utilizam mais que 100 clientes,

enquanto em um ambiente real, o número de clientes reportados é de milhares ou milhões. Os autores também constatam que mesmo esses trabalhos que utilizam mais de 100 clientes, geralmente usam clientes simulados e não dispositivos reais. Esse cenário, de acordo com o artigo, acontece devido as limitações dos *frameworks* existentes para implementar FL em ambientes reais e escaláveis, o que acaba limitando o desenvolvimento das pesquisas voltadas para esse contexto. Dito isso, Beutel et al. (2022) introduzem o Flower, um *framework* focado em oferecer suporte para execução de FL em dispositivos heterogêneos de forma escalável. Ele permite a implementação de um ambiente FL utilizando diferentes sistemas operacionais e hardware, e também oferece integração com diversas bibliotecas utilizadas para ML e com alguns algoritmos já existentes para FL, como FedAvg (MCMAHAN et al., 2016) e FedAvgM (HSU; QI; BROWN, 2019). Por fim, com o Flower foram executados experimentos utilizando até 15 milhões de clientes, com milhares selecionados em cada rodada de treinamento, cumprindo os objetivos do *framework* de possibilitar a execução de treinamentos FL em larga escala com um ou mais dispositivos, utilizando clientes heterogêneos podendo ser mesclados entre reais e simulados, e obtendo resultados em uma velocidade aceitável.

TensorFlow Federated (TFF) (GOOGLE, 2020) é um *software open-source* desenvolvido pela Google em 2019 para facilitar as pesquisas de FL. TFF utiliza Python e pode ser dividido em duas camadas: API Federated Core (FC), onde é implementada toda a lógica da computação federada baseado em uma linguagem interna para fazer as representações necessárias; e API Federated Learning, onde é oferecido um conjunto de interfaces de alto nível que possibilitam ao desenvolvedor utilizar a lógica federada implementada em FC. Por exemplo, são fornecidos decoradores de função Python que permitem definir uma função federada e executá-la em outro processo dentro de uma rede federada.

Com base nesse estudo da literatura foi possível observar que FL pode ser aplicado em diferentes áreas, compartilhando alguns desafios comuns e levantando outros específicos de cada área. A medida que avançamos no campo de pesquisa, cada vez mais FL será utilizado em ambientes reais e consequentemente surgirão vários problemas não identificados anteriormente. Dessa forma, além dos problemas já catalogados, é esperado que esse tema cresça exponencialmente nos próximos anos levantando muitos outros pontos para estudo e desenvolvimento.

3 Desenvolvimento

Devido ao caráter emergente de FL, frequentemente a documentação dos *frameworks* de FL é insuficiente e de difícil compreensão, principalmente para aqueles não familiarizados com ferramentas de ML, protocolos de rede e Python, que é a linguagem de programação mais empregada na ciência de dados. Em vista disso, neste capítulo serão descritos todos os processos feitos na parte prática desta pesquisa, a fim de documentar minuciosamente a escolha do *framework*, os procedimentos necessários para utilizá-lo e as ferramentas e tecnologias utilizadas. As bases de dados utilizadas nos testes são descritas na Seção 3.1 e o modelo construído descrito na Seção 3.2. Em seguida, na Seção 3.3, é explicado como foi a tomada de decisão em relação ao *framework* utilizado para realização dos experimentos com FL. Detalhes da implementação do servidor e cliente no *framework* escolhido se encontram nas Seções 3.4 e 3.5 respectivamente. Por fim, a Seção 3.6 aborda a utilização de dispositivos móveis como clientes, enumerando o que é necessário para essa implementação e pontuando as diferenças entre desenvolver um cliente para dispositivo móvel comparado ao exemplo apresentado na seção anterior.

3.1 Base de dados

Nesta seção são descritas as bases de dados utilizadas nos testes realizados nesta pesquisa. Ambas contêm imagens médicas relacionadas a doenças respiratórias. A Seção 3.1.1 aborda a base de dados utilizada na classificação de pneumonia, enquanto a Seção 3.1.2 trata da base de dados utilizada na classificação da COVID-19.

3.1.1 PneumoniaMNIST

A base de dados MedMNIST é um conjunto de dados de imagens médicas criado para fins de pesquisa na área de ML e está disponível publicamente para a comunidade científica. Foi construída no projeto (YANG et al., 2021) e é baseada no formato de dados MNIST, que é um conjunto de dados popular usado para reconhecimento de dígitos manuscritos. Contém 10 conjuntos de dados diferentes, cada um representando uma categoria diferente de imagem médica. As imagens foram coletadas de várias fontes, incluindo o National Institutes of Health, o Cancer Imaging Archive e outros bancos de imagens médicas. Além disso, todas as imagens foram redimensionadas para o tamanho 28x28 e convertidas em escala de cinza para garantir consistência entre os conjuntos de dados.

Para realização dos testes foi utilizado um dos conjuntos de dados contidos na base MedMNIST chamado PneumoniaMNIST 2D, representado pela Figura 3.1. Esse conjunto foi criado com o objetivo de ajudar a comunidade científica a desenvolver modelos mais precisos

para a detecção de pneumonia. Possui 5856 imagens de radiografia do tórax de crianças de 1 a 5 anos. Cada uma delas possui uma classificação binária indicando se o tórax foi afetado pela pneumonia ou não. O conjunto total foi dividido em 3 partes, resultando em 4708 imagens para treinamento, 524 para validação e 624 para teste.

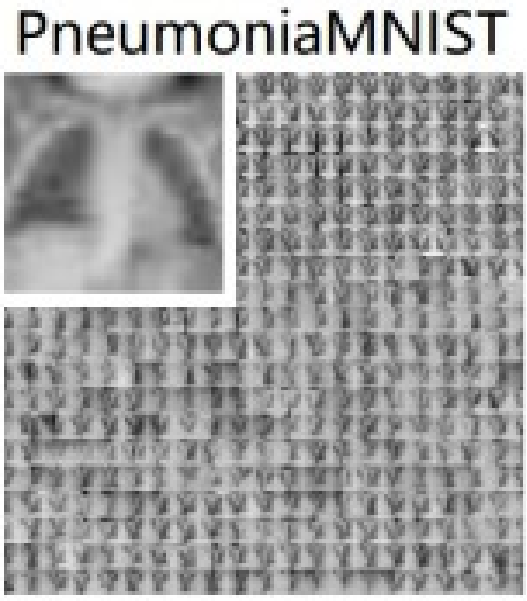


Figura 3.1 – Representação da base PneumoniaMNIST, contendo várias imagens de tórax em escala de cinza.
Fonte: (YANG et al., 2021).

3.1.2 COVID-19 *Radiography Database*

A base de dados de COVID-19 Rahman (2020) é uma coleção de imagens radiográficas de tórax de pacientes com COVID-19, pneumonia viral e outras doenças respiratórias. A base de dados contém mais de 15000 imagens radiográficas, incluindo 3616 imagens de COVID-19, 6012 de infecção pulmonar, 1345 imagens de pneumonia viral e 10192 imagens normais de tórax. Todas as imagens foram obtidas de diversas fontes públicas e privadas e são fornecidas em formato PNG com tamanho variando de 256 x 256 *pixels* a 2.328 x 2.952 *pixels*.

Essa base de dados é extremamente útil para pesquisas em ML e processamento de imagens, pois permite a criação e validação de algoritmos de diagnóstico de imagem para COVID-19. A base de dados também pode ser utilizada por especialistas em saúde para auxiliar no diagnóstico de pacientes com suspeita de COVID-19 e outras doenças respiratórias.

3.2 Modelos

Para cada base de dados foi utilizado um modelo diferente, dito isso, a Seção 3.2.1 descreve o modelo utilizado para a base de pneumonia 3.1.1 e a Seção 3.2.2 descreve o modelo utilizado para a base de COVID-19 3.1.2.

3.2.1 Modelo para PneumoniaMNIST

Foi criado um modelo a partir da interface *Sequential* fornecida pelo Keras (Código 3.1). Esse modelo se trata de uma Rede Neural Convolucional que espera uma única entrada e responde com uma única saída, ou seja, recebe uma imagem e devolve uma classificação. Foi construído desta forma por apresentar uma performance satisfatória e baixa complexidade para implementação. Possui 8 camadas, descritas a seguir:

1. Convolução utilizando 32 filtros, com ativação *ReLU*.
2. Convolução utilizando 64 filtros, com ativação *ReLU*.
3. *Max Pooling* para agrupar a imagem em conjuntos de 2 por 2 e escolher o maior valor, reduzindo o tamanho da imagem e mantendo as características que possuem maior valor.
4. *Dropout* para desativar 25% dos neurônios.
5. *Flatten* para converter o resultado do *Max Pooling* em um vetor.
6. Uma camada densa para conectar cada neurônio resultado do *Flatten*.
7. Mais um *Dropout* desativando 50% dos neurônios.
8. Por fim é adicionada mais uma camada densa com ativação *sigmoid* resultando em uma classificação binária.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28)))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation="sigmoid"))
```

Código 3.1 – Código utilizado para construir o modelo.

3.2.2 Modelo para COVID-19 *Radiography Database*

O modelo proposto no artigo Luz et al. (2021) foi construído a partir da rede neural EfficientNet para melhorar a performance e os resultados para o problema de classificação de COVID-19 em radiografias de tórax. A rede neural EfficientNet foi escolhida por apresentar uma boa performance e baixo custo computacional, visto que um dos objetivos de Luz et al. (2021) é propor um modelo eficiente que seja possível utilizar embarcado em dispositivos móveis. A Figura 3.2 ilustra a arquitetura do modelo, tendo como base as camadas da rede EfficientNet, conforme explicado.

Os resultados apresentados pelos experimentos realizados no artigo Luz et al. (2021) mostram que a abordagem proposta foi capaz de construir um modelo de alta qualidade para detectar COVID-19 em imagens de raio-x, com uma precisão geral de 93,9%. Além disso, o modelo apresentou sensibilidade de 96,8%, o que significa que ele conseguiu identificar a grande maioria dos casos positivos de COVID-19 na base de dados. A precisão do modelo também foi de 100%, o que indica que ele não cometeu erros ao classificar casos negativos como positivos. Outra informação relevante apresentada é que o modelo proposto utilizou de 5 a 30 vezes menos parâmetros do que outras arquiteturas testadas. Isso significa que ele conseguiu alcançar um desempenho semelhante ou superior ao de outros modelos utilizando menos recursos computacionais. Isso pode ser uma vantagem importante em aplicações práticas, onde a eficiência do modelo pode ser tão importante quanto sua precisão.

Stage	Operator	Resolution	#channels	#layers
1-9	EfficientNet B0	224x224	32	1
10	BN/Dropout	7x7	1280	1
11	FC/BN/Swich/Dropout	1	512	1
12	FC/BN/Swich	1	128	1
13	FC/Softmax	1	NC	1

Figura 3.2 – Arquitetura do modelo para COVID-19.
Fonte: (LUZ et al., 2021).

3.3 *Framework*

Para o desenvolvimento dessa pesquisa, a decisão mais importante foi a escolha da ferramenta para desenvolver FL. Para isso, foi procurada uma ferramenta que atenda aos seguintes pontos: manter a privacidade dos dados, possibilitar a comunicação entre computadores diferentes e não apenas simulações locais e possuir uma documentação que disponibilize a informação necessária sobre sua arquitetura.

O TFF foi o primeiro a ser analisado, com ele foi possível executar simulações locais com facilidade. Possui um tutorial com o procedimento necessário para criar servidor e cliente em máquinas diferentes e conectá-los. Nesse tutorial foi necessário executar a imagem do *runtime* no servidor e a imagem do *client* no cliente, ambas extraídas do repositório do TFF. Com essas aplicações rodando nas respectivas máquinas, foi possível executar uma simulação, porém houveram dificuldades para definir o fluxo de execução federado e utilizar os dados dos clientes localmente no treinamento. Dito isso, foi possível concluir que o TFF apresenta uma arquitetura complexa e uma documentação ainda em construção, o que não é o ideal para um pesquisa inicial como essa.

O segundo analisado foi o Flower. Ele possui uma arquitetura simples que encapsula a maior parte da lógica necessária para implementar servidor e clientes que se conectam e se comunicam em uma rede federada. Apresenta uma documentação coesa e de fácil entendimento, além de tutoriais para realizar testes com diversas bibliotecas oferecidas para implementação de ML. Por meio da documentação foi possível validar que as interfaces fornecidas para implementação de um cliente possibilitam o uso de dados locais, mantendo a privacidade. Portanto, atendendo a todos os requisitos, essa foi a ferramenta escolhida.

3.4 Servidor

Para criar o servidor o processo é simples, conforme mostrado no Código 3.2, sua função principal executa apenas três passos:

1. Obter e compilar um modelo. Por exemplo, o modelo construído descrito na Seção 3.2.
2. Inicializar o objeto *Strategy* passando como parâmetro as funções de treinamento e avaliação utilizando o modelo definido e também algumas configurações do treinamento, como número mínimo de clientes para cada etapa e os parâmetros iniciais do modelo.
3. Por fim, iniciar o servidor passando o endereço de IP desejado, o objeto *Strategy* e o número de rodadas de treinamento.

```
def main() -> None:
    model = utils.get_model()
    model.compile("adam", "binary_crossentropy", metrics=["accuracy"])

    strategy = fl.server.strategy.FedAvg(
        fraction_fit=0.2,
        fraction_evaluate=0.2,
        min_fit_clients=2,
        min_evaluate_clients=2,
        min_available_clients=2,
        evaluate_fn=get_evaluate_fn(model),
        on_fit_config_fn=fit_config,
        on_evaluate_config_fn=evaluate_config,
        initial_parameters=fl.common.ndarrays_to_parameters(model.get_weights()),
```

```
    )

    fl.server.start_server(
        server_address="0.0.0.0:8080",
        config=fl.server.ServerConfig(num_rounds=4),
        strategy=strategy,
    )
```

Código 3.2 – Função principal do servidor.

É necessário definir o número mínimo de clientes conectados ao servidor antes de uma rodada de treinamento começar, assim como um número mínimo de clientes a serem escolhidos para cada rodada de treinamento e avaliação. Após cada rodada, o modelo global é atualizado com os parâmetros resultados da última agregação feita e a acurácia é calculada executando a função de avaliação própria do modelo com base nos dados de teste. Todas essas configurações são passadas para o objeto *Strategy*. Por fim, ao iniciar o servidor é informado o número de rodadas de treinamento.

3.5 Cliente

No cliente é implementada uma classe herdando da interface *NumPyClient* fornecida pelo Flower (Código 3.3). Nessa classe, foram definidos um construtor e as funções de treinamento e avaliação. O construtor recebe um modelo e os dados de teste e treinamento separados entre imagens e rótulos; a função *fit* recebe os parâmetros e a configuração de treinamento informadas pelo servidor e treina o modelo local; e a função *evaluate* avalia os resultados com base nos dados de teste, retornando as porcentagens de perda e acurácia.

```
class Client(fl.client.NumPyClient):
    def __init__(self, model, x_train, y_train, x_test, y_test):
        self.model = model
        self.x_train, self.y_train = x_train, y_train
        self.x_test, self.y_test = x_test, y_test

    def fit(self, parameters, config):
        self.model.set_weights(parameters)
        batch_size: int = config["batch_size"]
        epochs: int = config["local_epochs"]
        history = self.model.fit(
            self.x_train,
            self.y_train,
            batch_size,
            epochs,
            validation_split=0.1,
        )
        parameters_prime = self.model.get_weights()
        num_examples_train = len(self.x_train)
        results = {
            "loss": history.history["loss"][0],
            "accuracy": history.history["accuracy"][0],
            "val_loss": history.history["val_loss"][0],
            "val_accuracy": history.history["val_accuracy"][0],
        }
```

```
26          return parameters_prime, num_examples_train, results
27
28      def evaluate(self, parameters, config):
29          self.model.set_weights(parameters)
30          steps: int = config["val_steps"]
31          local test data and return results
32          loss, accuracy = self.model.evaluate(self.x_test, self.y_test, 32, steps=steps)
33          num_examples_test = len(self.x_test)
34          return loss, num_examples_test, {"test_accuracy": accuracy}
```

Código 3.3 – Classe *Client* que implementa a interface fornecida pelo Flower *NumPyClient*.

Definida a classe *Client*, é necessária uma função *main* para executar o cliente (Código 3.4). Nessa função são executados os seguintes passos:

1. Obter e compilar um modelo. Por exemplo, o modelo construído descrito na Seção 3.2.
2. Obter os dados de teste e treino da base de dados. Por exemplo, as bases descritas na Seção 3.1.
3. Inicializar um objeto *Client* passando o modelo e os dados como parâmetro.
4. Por fim, iniciar o cliente passando o endereço de IP do servidor e o objeto *Client*.

```
1 def main() -> None:
2     model = utils.get_model()
3     model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
4     (x_train, y_train), (x_test, y_test) = utils.load_data()
5     client = Client(model, x_train, y_train, x_test, y_test)
6     fl.client.start_numpy_client(
7         server_address="localhost:8080",
8         client=client,
9     )
```

Código 3.4 – Função principal do cliente.

3.6 Dispositivos móveis

Resumidamente, o processo de criação de um treinamento com o Flower implica na definição do modelo de ML, configuração do servidor - incluindo o componente *Strategy* - configuração dos clientes e avaliação dos resultados. Ao seguir esses passos, é possível criar um sistema de treinamento federado utilizando o Flower.

Há diferenças significativas entre criar um treinamento com dispositivos móveis como clientes em comparação a computadores. A principal distinção é que para utilizar dispositivos móveis, o cliente e a *Strategy* precisam ser implementados de maneira personalizada para atender às necessidades específicas do sistema operacional do dispositivo.

Por exemplo, para implementar um treinamento com clientes Android, é necessário personalizar a serialização e deserialização dos dados de entrada e saída, para lidar com os dados

enviados pelos clientes Android para o servidor Python. A implementação da serialização e deserialização são feitas dentro do componente *Strategy*, um exemplo de implementação que funciona com o Android é apresentado no Código 3.5.

```
1
2  class FedAvgAndroid(Strategy):
3      def __init__(
4          self,
5          *,
6          fraction_fit: float = 1.0,
7          fraction_evaluate: float = 1.0,
8          min_fit_clients: int = 2,
9          min_evaluate_clients: int = 2,
10         min_available_clients: int = 2,
11         evaluate_fn: Optional[
12             Callable[
13                 [int, NDArrays, Dict[str, Scalar]],
14                 Optional[Tuple[float, Dict[str, Scalar]]],
15             ]
16         ] = None,
17         on_fit_config_fn: Optional[Callable[[int], Dict[str, Scalar]]] = None,
18         on_evaluate_config_fn: Optional[Callable[[int], Dict[str, Scalar]]] = None,
19         initial_parameters: Optional[Parameters] = None,
20     ) -> None:
21         super().__init__()
22         self.min_fit_clients = min_fit_clients
23         self.min_evaluate_clients = min_evaluate_clients
24         self.fraction_fit = fraction_fit
25         self.fraction_evaluate = fraction_evaluate
26         self.min_available_clients = min_available_clients
27         self.evaluate_fn = evaluate_fn
28         self.on_fit_config_fn = on_fit_config_fn
29         self.on_evaluate_config_fn = on_evaluate_config_fn
30         self.initial_parameters = initial_parameters
31
32     def evaluate(
33         self, server_round: int, parameters: Parameters
34     ) -> Optional[Tuple[float, Dict[str, Scalar]]]:
35         """Evaluate model parameters using an evaluation function."""
36         if self.evaluate_fn is None:
37             # No evaluation function provided
38             return None
39         weights = self.parameters_to_ndarrays(parameters)
40         eval_res = self.evaluate_fn(server_round, weights, {})
41         if eval_res is None:
42             return None
43         loss, metrics = eval_res
44         return loss, metrics
45
46     def configure_fit(
47         self, server_round: int, parameters: Parameters, client_manager: ClientManager
48     ) -> List[Tuple[ClientProxy, FitIns]]:
49         """Configure the next round of training."""
50         config = {}
51         if self.on_fit_config_fn is not None:
52             # Custom fit config function provided
53             config = self.on_fit_config_fn(server_round)
54         fit_ins = FitIns(parameters, config)
55
```

```
56          # Sample clients
57          sample_size, min_num_clients = self.num_fit_clients(
58              client_manager.num_available()
59          )
60          clients = client_manager.sample(
61              num_clients=sample_size, min_num_clients=min_num_clients
62          )
63
64          return [(client, fit_ins) for client in clients]
65
66      def configure_evaluate(
67          self, server_round: int, parameters: Parameters, client_manager: ClientManager
68      ) -> List[Tuple[ClientProxy, EvaluateIns]]:
69          """Configure the next round of evaluation."""
70
71          config = {}
72          if self.on_evaluate_config_fn is not None:
73              # Custom evaluation config function provided
74              config = self.on_evaluate_config_fn(server_round)
75          evaluate_ins = EvaluateIns(parameters, config)
76
77          # Sample clients
78          sample_size, min_num_clients = self.num_evaluation_clients(
79              client_manager.num_available()
80          )
81          clients = client_manager.sample(
82              num_clients=sample_size, min_num_clients=min_num_clients
83          )
84
85          return [(client, evaluate_ins) for client in clients]
86
87      def aggregate_fit(
88          self,
89          server_round: int,
90          results: List[Tuple[ClientProxy, FitRes]],
91          failures: List[Union[Tuple[ClientProxy, FitRes], BaseException]],
92      ) -> Tuple[Optional[Parameters], Dict[str, Scalar]]:
93          """Aggregate fit results using weighted average."""
94          if not results:
95              return None, {}
96          # Do not aggregate if there are failures and failures are not accepted
97          if not self.accept_failures and failures:
98              return None, {}
99          # Convert results
100         weights_results = [
101             (self.parameters_to_ndarrays(fit_res.parameters), fit_res.num_examples)
102             for client, fit_res in results
103         ]
104         return self.ndarrays_to_parameters(aggregate(weights_results)), {}
105
106     def aggregate_evaluate(
107         self,
108         server_round: int,
109         results: List[Tuple[ClientProxy, EvaluateRes]],
110         failures: List[Union[Tuple[ClientProxy, EvaluateRes], BaseException]],
111     ) -> Tuple[Optional[float], Dict[str, Scalar]]:
112         """Aggregate evaluation losses using weighted average."""
113         if not results:
114             return None, {}
115         # Do not aggregate if there are failures and failures are not accepted
```

```
116         if not self.accept_failures and failures:
117             return None, {}
118         loss_aggregated = weighted_loss_avg(
119             [
120                 (evaluate_res.num_examples, evaluate_res.loss)
121                 for _, evaluate_res in results
122             ]
123         )
124         return loss_aggregated, {}
125 
126     def ndarrays_to_parameters(self, ndarrays: NDArrays) -> Parameters:
127         tensors = [self.ndarray_to_bytes(ndarray) for ndarray in ndarrays]
128         return Parameters(tensors=tensors, tensor_type="numpy.nda")
129 
130     def parameters_to_ndarrays(self, parameters: Parameters) -> NDArrays:
131         return [self.bytes_to_ndarray(tensor) for tensor in parameters.tensors]
132 
133     def ndarray_to_bytes(self, ndarray: NDArray) -> bytes:
134         return ndarray.tobytes()
135 
136     def bytes_to_ndarray(self, tensor: bytes) -> NDArray:
137         ndarray_deserialized = np.frombuffer(tensor, dtype=np.float32)  # type: ignore
138         return cast(NDArray, ndarray_deserialized)
```

Código 3.5 – Implementação da Strategy FedAvg para Android.

Além disso, também é necessária a criação de um cliente personalizado que execute no dispositivo móvel. De acordo com o estudo realizado, isso pode ser feito de duas maneiras, (i) implementar um aplicativo e um cliente no dispositivo utilizando linguagens de programação compatíveis com cada tipo de dispositivo ou (ii) encontrar uma maneira de executar um cliente desenvolvido em Python no dispositivo. Ambas as abordagens apresentam grandes desafios, exigindo estudos aprofundados e trabalho de desenvolvimento para a construção de um treinamento personalizado usando dispositivos móveis.

3.7 VMs na *Google Cloud Plataform*

O provedor selecionado para a criação das VMs utilizadas nos referidos testes foi o GCP. Entretanto, vale mencionar que qualquer um dos provedores previamente mencionados poderia ter sido utilizado, tendo em vista que todos disponibilizam serviços gratuitos de criação de VMs por um período limitado.

Dentre os serviços oferecidos pelo Google Cloud Platform (GCP), o Google Compute Engine é um dos que viabiliza a criação de VMs. Para criar uma VM por meio desse serviço, basta seguir as instruções a seguir:

1. Na página inicial do Google Cloud, abra o menu lateral e seleciona a opção Compute Engine. Conforme mostrado na Figura 3.3 (a).

2. Na página inicial de Compute Engine, clique no botão "Criar instância". Conforme mostrado na Figura 3.3 (b).
3. No campo "Tipo de máquina", selecione uma máquina com o tamanho de RAM desejado. Conforme mostrado na Figura 3.3 (c).
4. No campo "Tamanho", selecione o tamanho de armazenamento desejado. Conforme mostrado na Figura 3.3 (d).
5. Caso necessário, habilite os tráfegos HTTP e HTTPS. Conforme mostrado na Figura 3.3 (e).
6. Crie a instância.

Para conduzir os testes realizados, foram criadas três máquinas virtuais, sendo uma delas designada como servidor e as outras duas, como clientes. Cabe salientar que, embora fossem utilizadas VMs, também seria possível utilizar máquinas físicas, desde que estas suportassem a execução de um terminal Unix. Ademais, é importante ressaltar que outros aspectos relevantes incluem a seleção adequada do tamanho da memória RAM e do CPU da máquina, bem como a habilitação de protocolos de regras de *firewall* (como HTTP e HTTPS). Essas decisões devem ser tomadas levando-se em consideração a natureza das atividades a serem executadas nas máquinas, sob pena de ocorrerem falhas por falta de recursos ou conectividade.

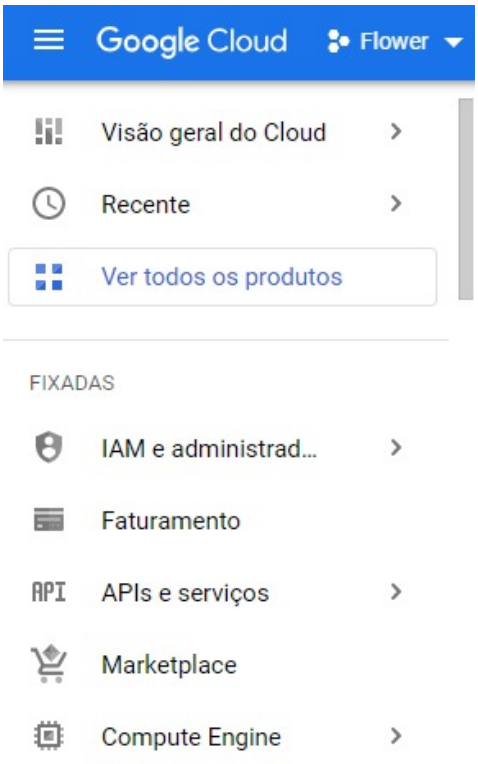
Para o experimento realizado com a base de dados PneumoniaMNIST e o modelo descrito na Seção 3.2, o tipo da máquina escolhido foi *e2-standard-2* (2 vCPU, 8 GB de memória). Porém, para o segundo experimento, realizado com a base de dados descrita na Seção 3.1.2, as máquinas mencionadas não foram suficientes, foi necessário criar máquinas que possuem mais recursos, o tipo escolhido foi *c2-standard-8* (8 vCPU, 32 GB de memória). Para todas as máquinas foi feita a habilitação dos protocolos HTTP e HTTPS.

Para permitir que outras máquinas se conectem à máquina do servidor, é imprescindível criar uma configuração de firewall de entrada que permita o acesso à porta na qual o servidor estará operando. A título de exemplo, caso a aplicação seja executada na porta 8080, é suficiente criar uma configuração que libere o protocolo TCP nesta porta.

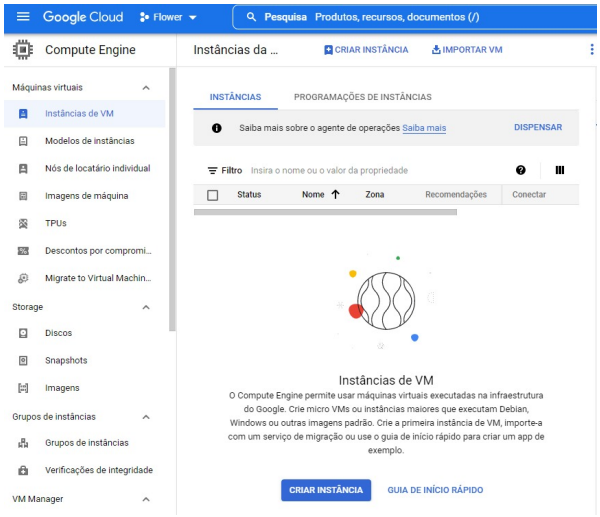
3.8 Jupyter Notebook

Ainda que o Google Colab seja muito poderoso e configurável junto ao Google Cloud, com ele foram encontradas algumas dificuldades para conectar o *notebook* nas VMs. Por esse motivo, para realização dos experimentos a ferramenta escolhida foi o Jupyter Notebook.

O Jupyter funciona como uma aplicação *web*, quando você o inicia, ele instancia uma aplicação no IP da máquina e na porta escolhida, isso torna possível acessá-lo por outro computador caso a conexão esteja liberada e a máquina esteja na mesma rede ou possua um endereço



(a) Menu lateral



(b) Seção Compute Engine

(c) Configuração da máquina

(d) Disco de inicialização

(e) Firewall

Figura 3.3 – Passo a passo para criação de uma VM no Google Cloud
Fonte: Elaborado pelo autor.

de IP único. Esse ponto foi o mais importante para sua escolha, porque permite a conexão com os *notebooks* executados nas VMs em uma máquina local através do navegador. A aplicação *web* consiste em uma interface para o diretório onde o Jupyter foi configurado, lá é possível criar e gerenciar notebooks, assim como outros arquivos presentes no diretório.

Para instalar o Jupyter Notebook e habilitar sua execução, execute os seguintes comandos:

```
python3 -m pip install jupyter
export PATH=$PATH:~/.local/bin
```

As instruções a seguir servem para criar um arquivo de configuração para o Jupyter e editá-lo incluindo a configuração que libera acessos de IPs externos a instância, permitindo, portanto, que outras máquinas acessem e executem o *notebook*:

```
jupyter notebook --generate-config
nano ~/.jupyter/jupyter_notebook_config.py
```

O último comando permite que você visualize e edite o arquivo de configuração gerado via terminal. Logo, para concluir a configuração, copie e cole o seguinte texto no terminal, substituindo o texto já existente no arquivo:

```
 c = get_config()
  c.NotebookApp.ip = '*'
  c.NotebookApp.open_browser = False
```

Para salvar o arquivo e fechá-lo, basta pressionar as teclas $CTRL + X$ e, em seguida, a tecla *Y*. Pronto, agora tudo está configurado, a execução do Jupyter pode ser feita por meio desse comando substituindo *PORT* pela porta escolhida:

```
jupyter notebook --port=PORT
```

4 Experimentos e Resultados

Neste capítulo são apresentados os detalhes e resultados dos experimentos realizados. Cada experimento foi executado 10 vezes e foi registrado o melhor valor, a média e o desvio padrão referentes as métricas acurácia, perda e tempo de execução. A Seção 4.1 descreve como foi a preparação para os experimentos com FL. Nas Seções 4.2 e 4.3 são encontrados os resultados dos testes que utilizam a base de dados PneumoniaMNIST e COVID-19 respectivamente. Por fim, na Seção 4.4 é feita uma análise e comparação dos resultados de cada experimento.

4.1 *Setup* para teste FL

Conforme descrito na Seção 3.7, foram criadas três máquinas virtuais para cada experimento utilizando o Google Cloud. Na máquina escolhida para ser o servidor foram seguidos os procedimentos descritos na Seção 3.4, com isso o servidor foi executado no endereço de IP da máquina na porta 8080. Nas outras duas máquinas foram executados os clientes construídos conforme descrito na Seção 3.5, substituindo apenas o campo *server_address* com o endereço de IP do servidor na porta 8080.

Foi definido um mínimo de dois clientes conectados ao servidor antes de uma rodada de treinamento começar. Esse mesmo número também se aplica ao mínimo de clientes escolhidos para treinamento e avaliação em cada rodada. Durante as rodadas um a três são executadas cinco etapas de avaliações locais em cada cliente, depois aumenta para dez etapas. Após cada rodada, o modelo global é atualizado com os parâmetros resultados da última agregação feita e a acurácia é calculada executando a função de avaliação própria do modelo com base nos dados de teste.

4.2 Testes com a base PneumoniaMNIST

Para ambos os testes (ML e FL) foi utilizado o modelo descrito na Seção 3.2, nesse modelo foram utilizados o otimizador *Adam* com *learning rate* igual a 0,001, a função de *loss binary crossentropy* e a métrica *accuracy*.

Para o teste com FL, a base de dados descrita na Seção 3.1.1 foi dividida entre os clientes, dessa forma um cliente utilizou a primeira metade e o outro cliente a última. Para o teste com ML foi utilizada a base toda.

A Tabela 4.1 apresenta os resultados dos testes com FL. O número de épocas globais pode ser visto como o número de rodadas de treinamento - a cada rodada os clientes executam o treinamento N vezes, onde N é igual ao número de épocas locais, em seguida os resultados são agregados. Dito isso, o número total de épocas pode ser obtido multiplicando o número de

rodadas por N.

Experimento	Épocas globais	Épocas locais	Total de épocas
1	2	10	20
2	4	5	20
3	10	2	20

Experimento	Acurácia (%)		Perda		Tempo (segundos)	
	Melhor	Média	Melhor	Média	Melhor	Média
1	85,90	82,95 ± 1,76	0,54	0,65 ± 0,08	129,47	164,44 ± 17,11
2	84,62	82,45 ± 1,73	0,60	0,78 ± 0,14	149,74	167,37 ± 10,48
3	86,70	83,94 ± 1,93	0,57	0,74 ± 0,13	187,06	201,04 ± 8,63

Tabela 4.1 – Resultados obtidos com FL e a base de dados PneumoniaMNIST.
Fonte: Elaborado pelo autor.

A Tabela 4.2 apresenta os resultados do teste com ML utilizando 20 épocas:

Acurácia (%)		Perda		Tempo (segundos)	
Melhor	Média	Melhor	Média	Melhor	Média
86,86	84,01 ± 2,15	0,46	0,79 ± 0,12	71,63	72,82 ± 0,71

Tabela 4.2 – Resultados obtidos com ML centralizado e a base de dados PneumoniaMNIST. Tempo em segundos.
Fonte: Elaborado pelo autor.

4.3 Testes com a base de COVID-19

Para os experimentos apresentados nesta seção foram utilizadas as mesmas configurações descritas no teste anterior, as únicas diferenças foram o modelo e a base de dados, além de ter sido necessário a utilização de máquinas com mais recursos, conforme descrito na Seção 3.7.

Foi utilizado o modelo descrito na Seção 3.2.2, nesse modelo foram utilizados o otimizador *Adam* com *learning rate* igual a 0,0001, a função de *loss categorical crossentropy* e a métrica *accuracy*.

Foram utilizadas 300 imagens de tórax com COVID-19 e 300 imagens de tórax saudáveis retiradas da base de dados descrita em 3.1.2 nos testes. No teste com FL, essas imagens foram divididas entre os dois clientes. A escolha de não utilizar todas as imagens disponíveis se deu por conta da falta de recursos computacionais para execução do treinamento com um volume muito grande de dados.

Experimento	Épocas globais	Épocas locais	Total de épocas
1	2	10	20
2	4	5	20
3	10	2	20

Experimento	Acurácia (%)		Perda		Tempo (segundos)	
	Melhor	Média	Melhor	Média	Melhor	Média
1	95,70	95,27 ± 0,51	0,13	0,14 ± 0,01	534,52	536,59 ± 1,65
2	95,91	95,30 ± 0,72	0,12	0,13 ± 0,01	688,26	690,03 ± 1,27
3	96,46	95,16 ± 0,94	0,11	0,14 ± 0,02	1.143,13	1.144,28 ± 1,27

Tabela 4.3 – Resultados obtidos com FL e a base de dados de COVID.
Fonte: Elaborado pelo autor.

A Tabela 4.4 apresenta os resultados do teste com ML utilizando 20 épocas, assim como os testes realizados na seção anterior.

Acurácia (%)		Perda		Tempo (segundos)	
Melhor	Média	Melhor	Média	Melhor	Média
95,39	95,02 ± 0,37	0,13	0,13 ± 0,01	371,95	372,67 ± 0,67

Tabela 4.4 – Resultados obtidos com ML tradicional e a base de dados COVID.
Fonte: Elaborado pelo autor.

4.4 Análise dos resultados

A partir dos resultados apresentados pela Tabela 4.1, é possível notar que o tempo de execução aumenta à medida que o número de épocas globais aumenta, devido ao aumento da comunicação entre clientes e servidor. Embora os experimentos apresentem resultados semelhantes em relação à acurácia e perda, os experimentos 1 e 3 obtiveram resultados um pouco melhores do que o experimento 2, indicando que o balanceamento de épocas locais e globais pode ser fundamental para obter melhores resultados.

A comparação entre os resultados das Tabelas 4.1 e 4.2 mostra que o tempo de execução dos testes com FL é praticamente o dobro dos testes com ML, embora ambos obtenham resultados semelhantes em relação à acurácia e perda.

Os valores de acurácia e perda dos experimentos apresentados na Tabela 4.3 são muito próximos, variando ainda menos que os resultados da Tabela 4.1 analisados previamente, provavelmente devido à maior complexidade do modelo utilizado. Observou-se também uma relação diretamente proporcional entre o número de épocas globais e o tempo de processamento, reforçando o quanto pode ser custoso as comunicações entre cliente e servidor.

As Tabelas 4.3 e 4.4 também apresentam resultados semelhantes em relação à acurácia e perda, bem como um aumento considerável no tempo de execução nos testes com FL, em comparação aos testes com ML. Para realizar uma análise mais concreta, seria necessário a

realização de testes adaptados para simular condições reais de treinamento, como aumentar o número de clientes e desbalancear a distribuição dos dados de maneira quantitativa e qualitativa.

5 Considerações Finais

Neste capítulo é feito um retrospecto dos objetivos que foram possíveis concluir e da contribuição dessa pesquisa para o meio acadêmico, além de pontuar trabalhos que serão realizados no futuro.

5.1 Conclusão

Por meio do estudo realizado nesta monografia, foi possível descrever características, aplicações e desafios do *Federated Learning*, assim como outros temas relacionados.

No desenvolvimento, um dos principais desafios foi a construção do ambiente FL, visto que por ser uma área muito emergente grande parte dos *frameworks* ainda estão em construção e possuem uma documentação incompleta. Após uma investigação minuciosa, foi identificado um *framework* promissor (FLOWER, 2019), que já se encontra em uma versão mais avançada, e por meio dele foi possível realizar testes utilizando um servidor e dois clientes, executados em diferentes computadores, sem comprometer a privacidade dos dados, uma vez que cada cliente utilizou dados locais para o treinamento. Com o auxílio do Flower, foram realizados experimentos com modelos e bases de dados distintas, bem como testes com ML centralizado, configurados de forma equivalente aos testes realizados com o FL. Dessa forma, tornou-se possível realizar uma breve análise comparativa entre os resultados obtidos em termos de acurácia, perda e tempo de execução nos testes com FL e ML.

Como resultado da elaboração dos testes mencionados nos parágrafos anteriores, foi criada uma base de conhecimento sobre o desenvolvimento de um ambiente FL e as ferramentas e tecnologias disponíveis para facilitar essa tarefa. Ademais, um tutorial (Apêndice A) simplificado foi implementado para demonstrar como realizar um treinamento FL utilizando o Flower e o conjunto de dados PneumoniaMNIST. É relevante enfatizar a importância da criação de conteúdos desse tipo para tornar o *Federated Learning* uma tecnologia mais acessível aos iniciantes na área.

Desta forma, esta pesquisa oferece três importantes contribuições: (i) fundamentação teórica sobre o FL e temas correlatos; (ii) ilustração da implementação prática de um treinamento federado utilizando a plataforma Flower; (iii) uma base de conhecimento sobre as ferramentas e tecnologias empregadas nos experimentos de FL, incluindo a documentação das dificuldades e obstáculos encontrados, de modo que desenvolvedores futuros possam abordar o tema com maior facilidade.

5.2 Trabalhos Futuros

Nesta pesquisa foi possível realizar testes simples com a ferramenta Flower e fazer um paralelo com os resultados do teste utilizando ML tradicional. No entanto, esses testes foram feitos sem variação de número de clientes e distribuição da base de dados. Para dar continuidade a esses experimentos, poderiam ser realizados treinamentos explorando ao máximo essas variáveis, a fim de identificar as nuances do treinamento FL em situações mais fiéis a realidade e com isso aprofundar mais nas análises e discussões acerca dos testes.

Além disso, foram realizados testes em dispositivos móveis, embora não tenha sido possível obter resultados expressivos ou explorar muito essa vertente de teste, devido à complexidade de implementar o cliente de forma personalizada para executar em um aplicativo nativo de cada tipo de dispositivo ou executar um cliente implementado em Python nesses dispositivos. Portanto, um trabalho futuro seria aprofundar as investigações utilizando dispositivos móveis como clientes, já que essa é uma das principais aplicações de FL e requer uma análise mais detalhada para o desenvolvimento e realização de treinamentos.

Referências

AGRAWAL, S.; SARKAR, S.; AOUEDI, O.; YENDURI, G.; PIAMRAT, K.; BHATTACHARYA, S.; MADDIKUNTA, P. K. R.; GADEKALLU, T. R. Federated learning for intrusion detection system: Concepts, challenges and future directions. *CoRR*, abs/2106.09527, 2021. Disponível em: <https://arxiv.org/abs/2106.09527>.

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: IEEE. *2017 international conference on engineering and technology (ICET)*. [S.l.], 2017. p. 1–6.

ALEDHARI, M.; RAZZAK, R.; PARIZI, R. M.; SAEED, F. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, v. 8, p. 140699–140725, 2020.

ALZUBAIDI, L.; ZHANG, J.; HUMAIDI, A. J.; AL-DUJAILI, A.; DUAN, Y.; AL-SHAMMA, O.; SANTAMARÍA, J.; FADHEL, M. A.; AL-AMIDIE, M.; FARHAN, L. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, Springer, v. 8, n. 1, p. 1–74, 2021.

BEUTEL, D. J.; TOPAL, T.; MATHUR, A.; QIU, X.; FERNANDEZ-MARQUES, J.; GAO, Y.; SANI, L.; LI, K. H.; PARCOLLET, T.; GUSMÃO, P. P. B. de et al. Flower: A friendly federated learning framework. 2022.

CECCON, D. *Funções de ativação: definição, características, e quando usar cada uma*. 2020. <https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma/>.

CHEN, Y.-T.; CHUANG, Y.-C.; WU, A.-Y. Online extreme learning machine design for the application of federated learning. In: *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. [S.l.: s.n.], 2020. p. 188–192.

DICK, S. Artificial intelligence. PubPub, 2019.

DIGEST. *Centralized Learning vs. Distributed Learning*. 2021. <https://digestize.medium.com/centralized-learning-vs-distributed-learning-c75ee9e94423>.

DU, Z.; WU, C.; YOSHINAGA, T.; YAU, K.-L. A.; JI, Y.; LI, J. Federated learning for vehicular internet of things: Recent advances and open issues. *IEEE Open Journal of the Computer Society*, IEEE, v. 1, p. 45–61, 2020.

FLOWER. *Flower: A Friendly Federated Learning Framework*. 2019. <https://flower.dev/>.

FOUNDATION, T. A. S. *Apache Hadoop*. 2006–2023. <https://hadoop.apache.org/>.

FOUNDATION, T. A. S. *Apache Spark*. 2006–2023. <https://spark.apache.org/>.

GOOGLE. *Tensorflow federated: Machine learning on decentralized data*. 2020. <https://www.tensorflow.org/federated>.

GRILL-SPECTOR, K.; KANWISHER, N. Visual recognition: As soon as you know it is there, you know what it is. *Psychological Science*, SAGE Publications Sage CA: Los Angeles, CA, v. 16, n. 2, p. 152–160, 2005.

HSU, T. H.; QI, H.; BROWN, M. Measuring the effects of non-identical data distribution for federated visual classification. *CoRR*, abs/1909.06335, 2019. Disponível em: <http://arxiv.org/abs/1909.06335>.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. *Science*, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015.

KAIROUZ, P.; MCMAHAN, H. B.; AVENT, B.; BELLET, A.; BENNIS, M.; BHAGOJI, A. N.; BONAWITZ, K. A.; CHARLES, Z.; CORMODE, G.; CUMMINGS, R.; D'OLIVEIRA, R. G. L.; ROUAYHEB, S. E.; EVANS, D.; GARDNER, J.; GARRETT, Z.; GASCÓN, A.; GHAZI, B.; GIBBONS, P. B.; GRUTESER, M.; HARCHAOUI, Z.; HE, C.; HE, L.; HUO, Z.; HUTCHINSON, B.; HSU, J.; JAGGI, M.; JAVIDI, T.; JOSHI, G.; KHODAK, M.; KONEČNÝ, J.; KOROLOVA, A.; KOUSHANFAR, F.; KOYEJO, S.; LEPOINT, T.; LIU, Y.; MITTAL, P.; MOHRI, M.; NOCK, R.; ÖZGÜR, A.; PAGH, R.; RAYKOVA, M.; QI, H.; RAMAGE, D.; RASKAR, R.; SONG, D.; SONG, W.; STICH, S. U.; SUN, Z.; SURESH, A. T.; TRAMÈR, F.; VEPAKOMMA, P.; WANG, J.; XIONG, L.; XU, Z.; YANG, Q.; YU, F. X.; YU, H.; ZHAO, S. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019. Disponível em: <http://arxiv.org/abs/1912.04977>.

KAIROUZ, P.; MCMAHAN, H. B.; AVENT, B.; BELLET, A.; BENNIS, M.; BHAGOJI, A. N.; BONAWITZ, K.; CHARLES, Z.; CORMODE, G.; CUMMINGS, R. et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, Now Publishers, Inc., v. 14, n. 1–2, p. 1–210, 2021.

KIM, H.; PARK, J.; BENNIS, M.; KIM, S.-L. Blockchained on-device federated learning. *IEEE Communications Letters*, IEEE, v. 24, n. 6, p. 1279–1283, 2019.

KIUKKONEN, N.; BLOM, J.; DOUSSE, O.; GATICA-PEREZ, D.; LAURILA, J. Towards rich mobile phone datasets: Lausanne data collection campaign. *Proc. ICPS, Berlin*, v. 68, n. 7, 2010.

LANG, N. *Using Convolutional Neural Network for Image Classification*. 2021. <https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4>.

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

LI, T.; SAHU, A. K.; ZAHEER, M.; SANJABI, M.; TALWALKAR, A.; SMITH, V. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, v. 2, p. 429–450, 2020.

LUZ, E.; SILVA, P.; SILVA, R.; SILVA, L.; GUIMARÃES, J.; MIOZZO, G.; MOREIRA, G.; MENOTTI, D. Towards an effective and efficient deep learning model for COVID-19 patterns detection in x-ray images. *Research on Biomedical Engineering*, Springer Science and Business Media LLC, v. 38, n. 1, p. 149–162, apr 2021. Disponível em: <https://doi.org/10.1007\%2Fs42600-021-00151-6>.

MATHUR, A.; BEUTEL, D. J.; GUSMãO, P. P. B. de; FERNANDEZ-MARQUES, J.; TOPAL, T.; QIU, X.; PARCOLLET, T.; GAO, Y.; LANE, N. D. On-device federated learning with flower. arXiv, 2021. Disponível em: <https://arxiv.org/abs/2104.03042>.

MCMAHAN, B.; RAMAGE, D. *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. 2017. <https://ai.googleblog.com/2017/04/federated-learning-collaborative.html>.

MCMAHAN, H. B.; MOORE, E.; RAMAGE, D.; HAMPSON, S.; ARCAS, B. A. y. Communication-efficient learning of deep networks from decentralized data. arXiv, 2016. Disponível em: <https://arxiv.org/abs/1602.05629>.

MITCHELL, T. M.; MITCHELL, T. M. *Machine learning*. [S.l.]: McGraw-hill New York, 1997. v. 1.

NGUYEN, D. C.; DING, M.; PATHIRANA, P. N.; SENEVIRATNE, A.; LI, J.; NIYATO, D.; POOR, H. V. Federated learning for industrial internet of things in future industries. *IEEE Wireless Communications*, IEEE, 2021.

ORACLE. *What is Big Data?* 2022. <https://www.oracle.com/big-data/what-is-big-data/#:~:text=The%20definition%20of%20big%20data,especially%20from%20new%20data%20sources/>.

PEIXOTO, L. H. R. *Aprendizado de Máquina Aplicado no Atendimento de Reclamações de Clientes*. Tese (Doutorado) — Universidade de São Paulo, 2021.

PHILLIPS-WREN, G. Ai tools in decision making support systems: a review. *International Journal on Artificial Intelligence Tools*, World Scientific, v. 21, n. 02, p. 1240005, 2012.

RAHMAN, T. *COVID-19 Radiography Database*. [S.l.]: Kaggle, 2020. <https://www.kaggle.com/tawsifurrahman/covid19-radiography-database>.

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. The future of digital health with federated learning. *NPJ digital medicine*, Nature Publishing Group, v. 3, n. 1, p. 1–7, 2020.

RYAN, L. *gRPC Motivation and Design Principles*. 2015. <https://grpc.io/blog/principles/>.

SHEN, S.; JIANG, H.; ZHANG, T. Stock market forecasting using machine learning algorithms. *Department of Electrical Engineering, Stanford University, Stanford, CA*, Citeseer, p. 1–5, 2012.

SILVA, L. N. M. *Tipos de aprendizado de máquina e algumas aplicações*. 2021. <http://www2.decom.ufop.br/terralab/tipos-de-aprendizado-de-maquina-e-algumas-aplicacoes/>.

SUNNY, B. K.; JANARDHANAN, P. S.; FRANCIS, A. B.; MURALI, R. Implementation of a self-adaptive real time recommendation system using spark machine learning libraries. In: *2017 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*. [S.l.: s.n.], 2017. p. 1–7.

THENNAKOON, A.; BHAGYANI, C.; PREMADASA, S.; MIHIRANGA, S.; KURUWITAARACHCHI, N. Real-time credit card fraud detection using machine learning. In: IEEE. *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. [S.l.], 2019. p. 488–493.

VERBRAEKEN, J.; WOLTING, M.; KATZY, J.; KLOPPENBURG, J.; VERBELEN, T.; RELLERMEYER, J. S. A survey on distributed machine learning. *Acm computing surveys (csur)*, ACM New York, NY, USA, v. 53, n. 2, p. 1–33, 2020.

WANG, K.; BABENKO, B.; BELONGIE, S. End-to-end scene text recognition. In: IEEE. *2011 International conference on computer vision*. [S.l.], 2011. p. 1457–1464.

YAMASHITA, R.; NISHIO, M.; DO, R. K. G.; TOGASHI, K. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, Springer, v. 9, n. 4, p. 611–629, 2018.

YANG, J.; SHI, R.; WEI, D.; LIU, Z.; ZHAO, L.; KE, B.; PFISTER, H.; NI, B. Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification. *arXiv preprint arXiv:2110.14795*, 2021.

ZANTALIS, F.; KOULOURAS, G.; KARABETSOS, S.; KANDRIS, D. A review of machine learning and iot in smart transportation. *Future Internet*, Multidisciplinary Digital Publishing Institute, v. 11, n. 4, p. 94, 2019.

Apêndices

APÊNDICE A – Tutorial com Notebook Python

Este apêndice documenta o *notebook* construído em formato de tutorial para execução de um treinamento FL utilizando o Flower e a base de dados PneumoniaMNIST.

Aplicação de Federated Learning (FL) utilizando Google Cloud, Flower e MEDMNIST

Nesse tutorial rodaremos um treinamento simples de FL utilizando o framework Flower, a base de dados MEDMNIST e o Google Cloud para criação de três máquinas virtuais, duas que serão utilizadas como clientes e uma como servidor. Para maior conhecimento é indicado a seguinte leitura: https://flower.dev/docs/tutorial/Flower-0-What-is-FL.html.

Base de dados

PneumoniaMNIST: https://medmnist.com/ Como obter os dados: https://github.com/MedMNIST/MedMNIST/blob/main/examples/getting_started_without_PyTorch.ipynb

Configurações utilizadas

120GB Armazenamento 8GB Ram SO: debian-11-bullseye-v20230206 Regras de firewall (entrada):

- HTTP - HTTPs
- tcp 5000 - Conectar jupyter notebook
- tcp 8080 - Acesso ao servidor

Tutorial

É necessário executar os seguintes passos em todas as máquinas. Para começar instale o git e clone o repositório desse projeto:

- sudo apt-get install git
- git clone https://github.com/thborba/federated-learning.git

Faça a instalação do jupyter notebook:

- sudo apt-get install python3-pip
- python3 -m pip install jupyter
- export PATH=$PATH:~/.local/bin

A configuração a seguir é necessária para acessar o notebook de outro computador:

- jupyter notebook --generate-config
- nano ~/.jupyter/jupyter_notebook_config.py

  Copiar e colar seguinte texto:

```
c = get_config()
c.NotebookApp.ip = '*'
c.NotebookApp.open_browser = False
```

CTRL + X -> Y para salvar

Execute o jupyter:

- jupyter notebook --port=5000

Instalação de dependências

```
!python3 -m pip install -r requirements.txt
```

Executar Servidor

```
from typing import Dict, Optional, Tuple
import flwr as fl
import utils

# [::] is used to expose the server both on internal and external ip
SERVER_ADDRESS = "[::]:8080"

def main() -> None:
    model = utils.get_model()
    model.compile("adam", "binary_crossentropy", metrics=["accuracy"])

    strategy = fl.server.strategy.FedAvg(
        fraction_fit=0.2,
        fraction_evaluate=0.2,
        min_fit_clients=2,
        min_evaluate_clients=2,
        min_available_clients=2,
        evaluate_fn=get_evaluate_fn(model),
        on_fit_config_fn=fit_config,
        on_evaluate_config_fn=evaluate_config,

initial_parameters=fl.common.ndarrays_to_parameters(model.get_weights(
)),
    )

    fl.server.start_server(
        server_address="[::]:8080",
        config=fl.server.ServerConfig(num_rounds=4),
        strategy=strategy,
    )


def get_evaluate_fn(model):
    """Return an evaluation function for server-side evaluation."""
    _, (x_test, y_test) = utils.load_data()
    # The `evaluate` function will be called after every round
```

```
    def evaluate(
        parameters: fl.common.NDArrays,
    ) -> Optional[Tuple[float, Dict[str, fl.common.Scalar]]]:
        model.set_weights(parameters)  # Update model with the latest
parameters
        loss, accuracy = model.evaluate(x_test, y_test)
        return loss, {"test_accuracy": accuracy}


    return evaluate


def fit_config(rnd: int):
    """Return training configuration dict for each round."""
    config = {
        "batch_size": 32,
        "local_epochs": 1 if rnd < 2 else 4,
    }
    return config


def evaluate_config(rnd: int):
    """Return evaluation configuration dict for each round."""
    val_steps = 5 if rnd < 4 else 10
    return {"val_steps": val_steps}

main()
```

Executar Cliente

```
import os
import utils
import flwr as fl

SERVER_ADDRESS = "35.234.149.156:8080"
CLIENT_NUMBER = 0

# Make TensorFlow logs less verbose
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

# Define Flower client
class Client(fl.client.NumPyClient):
    def __init__(self, model, x_train, y_train, x_test, y_test):
        self.model = model
        self.x_train, self.y_train = x_train, y_train
        self.x_test, self.y_test = x_test, y_test

    def fit(self, parameters, config):
        """Train parameters on the locally held training set."""
```

```
        # Update local model parameters
        self.model.set_weights(parameters)

        # Get hyperparameters for this round
        batch_size: int = config["batch_size"]
        epochs: int = config["local_epochs"]

        # Train the model using hyperparameters from config
        history = self.model.fit(
            self.x_train,
            self.y_train,
            batch_size,
            epochs,
            validation_split=0.1,
        )

        # Return updated model parameters and results
        parameters_prime = self.model.get_weights()
        num_examples_train = len(self.x_train)
        results = {
            "loss": history.history["loss"][0],
            "accuracy": history.history["accuracy"][0],
            "val_loss": history.history["val_loss"][0],
            "val_accuracy": history.history["val_accuracy"][0],
        }
        return parameters_prime, num_examples_train, results

    def evaluate(self, parameters, config):
        """Evaluate parameters on the locally held test set."""

        # Update local model with global parameters
        self.model.set_weights(parameters)

        # Get config values
        steps: int = config["val_steps"]

        # Evaluate global model parameters on the local test data and
return results
        loss, accuracy = self.model.evaluate(self.x_test, self.y_test,
32, steps=steps)
        num_examples_test = len(self.x_test)
        return loss, num_examples_test, {"test_accuracy": accuracy}


def main() -> None:
    model = utils.get_model()
    model.compile("adam", "binary_crossentropy", metrics=["accuracy"])

    (x_train, y_train), (x_test, y_test) =
```

```
utils.load_data(CLIENT_NUMBER)
    client = Client(model, x_train, y_train, x_test, y_test)

    fl.client.start_numpy_client(
        server_address=SERVER_ADDRESS,
        client=client,
    )

main()
```