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

LUCAS DE SOUZA SILVA
Orientador: Dr. Pedro Henrique Lopes Silva
Coorientador: Dr. Joubert de Castro Lima

ANÁLISE E IMPLANTAÇÃO DE FERRAMENTAS PARA APRENDIZADO FEDERADO LOCAL E EM NUVEM: UM BREVE COMPARATIVO ENTRE O TREINAMENTO LOCAL E FEDERADO

Ouro Preto, MG
2023

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

LUCAS DE SOUZA SILVA

ANÁLISE E IMPLANTAÇÃO DE FERRAMENTAS PARA APRENDIZADO FEDERADO LOCAL E EM NUVEM: UM BREVE COMPARATIVO ENTRE O TREINAMENTO LOCAL E FEDERADO

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
2023

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

FOLHA DE APROVAÇÃO

Lucas de Souza Silva

Análise e implantação de ferramentas para Aprendizado Federado local e em nuvem: Um breve comparativo entre o treinamento local e federado

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
Andrea Gomes Campos Bianchi (Examinadora) - Doutora - 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 22:35, 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 **0493051** e o código CRC **4E2580A2**.

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

Dedico este trabalho aos meus queridos familiares e amigos, que sempre estiveram ao meu lado durante toda a minha jornada acadêmica.

A vocês, que me apoiaram e me encorajaram a seguir em frente mesmo diante dos obstáculos, deixo meu eterno agradecimento.

Sem o amor, a paciência, o incentivo e o carinho de vocês, eu não teria chegado até aqui. Cada uma das palavras de estímulo, cada abraço, cada sorriso foram fundamentais para que eu pudesse superar as dificuldades e alcançar meus objetivos.

Portanto, este trabalho é dedicado a vocês com muito carinho e gratidão. Que possamos seguir juntos, sempre, comemorando cada conquista e enfrentando juntos os desafios que a vida nos apresenta.

Agradecimentos

Gostaria de expressar minha gratidão a todos que contribuíram para a realização deste trabalho. Em especial, gostaria de agradecer aos meus orientadores, Dr. Pedro Henrique Lopes Silva, Dr. Joubert de Castro Lima e Dr. Rodrigo César Pedrosa Silva, pela dedicação e paciência ao longo deste processo. Seus conselhos e orientações foram essenciais para que este trabalho pudesse ser concluído.

Também gostaria de agradecer aos professores que me ensinaram ao longo do curso e que contribuíram para o meu desenvolvimento acadêmico. Agradeço aos amigos de trabalho que me apoiaram, encorajaram e colaboraram com suas experiências e conhecimentos.

Por fim, agradeço à minha família por sempre me apoiar em todas as etapas da minha vida acadêmica, e a todos que de alguma forma contribuíram para que este trabalho fosse concluído. Obrigado!

Resumo

O aprendizado federado consiste em uma forma de executar o aprendizado de máquina distribuído sem que o cientista de dados precise conhecer ou obter todos os dados localmente. Essa forma de treinamento envolve não só a área de inteligência artificial, mas também a área de sistemas distribuídos.

Neste trabalho, foram exploradas ferramentas disponíveis no mercado para analisar a facilidade de implantação e utilização do aprendizado federado, bem como comparativos breves entre os métodos convencionais e federado. Os resultados mostraram que o aprendizado federado pode ser facilmente utilizado sem que o cientista precise obter conhecimento além do utilizado no treinamento local.

Na segunda parte do trabalho, foi dada ênfase à automatização dos procedimentos necessários para a execução do treinamento federado. O objetivo da automação é facilitar o uso e acelerar a curva de aprendizado da ferramenta desenvolvida pela IBM. Além disso, novos experimentos foram realizados com variações na quantidade e proporção dos dados, a fim de analisar como o aprendizado federado lida com essas variáveis.

Os resultados obtidos mostram que, dependendo da base de dados analisada, o viés criado pode ser benéfico para a eficiência do treinamento. Para a base de dados apresentada, que é binária, a divisão por classe mostrou-se vantajosa à medida que se aumenta a quantidade de épocas e participantes. Além disso, a criação de um tutorial facilitou significativamente todo o processo, desde a preparação até a execução do treinamento, reduzindo assim o tempo necessário para realizar cada etapa do processo de treinamento.

Palavras-chave: *Federated learning*. *Machine Learning*. Sistemas Distribuídos. *PySyft*. *IBM Federated Learning*.

Abstract

Federated learning is a way to execute distributed machine learning without requiring the data scientist to have access to all the data locally. This training approach involves not only the field of artificial intelligence but also distributed systems.

In this work, we explored available tools in the market to analyze the ease of implementation and usage of federated learning, as well as brief comparisons between conventional and federated methods. The results showed that federated learning can be easily used without the data scientist needing knowledge beyond that used in local training.

In the second part of the work, we focused on automating the procedures necessary for the execution of federated training. The automation aims to facilitate usage and accelerate the learning curve of the IBM-developed tool. In addition, new trainings were carried out with variations in the amount and proportion of data to analyze how federated learning deals with these variables.

The results obtained show that, depending on the analyzed dataset, the bias created can be beneficial to the training efficiency. For the presented binary dataset, class division proved to be advantageous as the number of epochs and participants increased. Furthermore, the creation of a tutorial significantly facilitated the entire process, from preparation to training execution, reducing the time required to perform each stage of the training process.

Keywords: Federated learning. Machine Learning. Distributed systems. PySyft. IBM Federated Learning.

Lista de Ilustrações

Lista de Tabelas

Lista de Códigos Fonte

Lista de Abreviaturas e Siglas

API	*Application Programming Interface*
CLI	*Command Line Interface*
CPU	*Central Processing Unit*
DECOM	Departamento de Computação
FE	*Functional Encryption*
FHE/SHE	*Homomorphic Encryption*
IA	Inteligência Artificial
IP	*Internet Protocol*
ML	*Machine Learning*
PD	Privacidade Diferencial
SGD	*Stochastic Gradient Descent*
SMPC	*Secure Multi-Party Computation*
UFOP	Universidade Federal de Ouro Preto

Sumário

1 Introdução

É frequente que a mídia noticie escândalos relacionados a vazamentos de dados, como o caso que ocorreu com o *Facebook* (BBC, 2018). Grande parte desses dados vazados advém de grandes bases de dados que possuem esquemas de segurança, o que demonstra a fragilidade dos nossos dados e o quão expostos podemos estar. Nesse sentido, o avanço constante da tecnologia traz consigo os perigos do uso indevido da mesma. Atualmente, tem-se observado o crescente uso da aprendizagem de máquina para alcançar resultados satisfatórios em áreas que antes eram pouco exploradas, como saúde, finanças, antifraude, entre outras. O que essas áreas têm em comum é a necessidade de proteger os dados utilizados no treinamento.

Com base nos estudos mais recentes, muito tem se falado sobre a ampliação da área de *Machine Learning* com a aplicação do mesmo juntamente com Sistemas Distribuídos, no que está sendo chamado de *Federated Learning* (que em tradução literal significa Aprendizado Federado) (Cem Dilmegani, 2022). O *Federated Learning* é um campo relativamente novo no mundo de *Machine Learning*, cujo objetivo é realizar o aprendizado de máquina de forma descentralizada, utilizando diversos dispositivos que possuem os dados de treinamento armazenados localmente (Kim Martineau, 2022). Posteriormente, os dados referentes aos treinamentos são reunidos no intuito de melhorar o algoritmo, calculando uma média entre os resultados obtidos, para que novas rodadas de treinamento sejam realizadas a partir da média dos treinamentos anteriores.

O Aprendizado Federado vem sendo estudado para uso em ambientes cujo objeto de pesquisa possui algum nível de restrição quanto à sua manipulação ou divulgação (Nasron Cheong, 2022). Dessa forma, os mesmos não podem ser enviados para a nuvem ou para fora das organizações que os detêm, e por esse motivo, o algoritmo deve ir até os dados para a realização dos treinamentos. Além de realizar os treinamentos de forma remota, o Aprendizado Federado ainda pode fazer uso de técnicas como *Secure Multi-Party Computation* (SMPC), *Homomorphic Encryption* (FHE/SHE) e *Functional Encryption* (FE) para realizar o treinamento sobre valores criptografados. Aumentando, assim, a segurança sobre os dados trabalhados.

Segundo Budrionis et al. (2021), muitos estudos na área da saúde tiveram dificuldades para evoluir devido às barreiras de acesso a informações hospitalares, por possuírem uma base de dados reduzida para a realização dos projetos. Neste caso, o Aprendizado Federado surge como uma alternativa para os problemas enfrentados, por manter a privacidade dos dados, disponibilizar mais materiais de amostra e aumentar o alcance da rede de pesquisa. Visto isso, a ferramenta em questão proporciona treinamentos mais efetivos para o Machine Learning. Em vista dos desafios citados, o Federated Learning se mostra favorável para a resolução das barreiras enfrentadas pelos métodos tradicionais de aprendizado.

Dessa forma, o Cientista de Dados não precisará ter conhecimento avançado em distri-

buição de treinamento, tornando o processo de desenvolvimento mais acessível. A ferramenta escolhida para ilustrar essa abordagem é a biblioteca de código aberto denominada *IBM fl*, que permite ao usuário realizar o *Deep Learning* de forma segura, privada e distribuída, utilizando-se de extensões de bibliotecas renomadas como *PyTorch*, *Keras* e *Tensorflow*.

Neste estudo, será utilizada a base de dados PneumoniaMnist (ver seção 3.1) no intuito de gerar uma estrutura simples e de fácil execução para o treinamento federado, além de realizar experimentos com o objetivo de entender os impactos de uma distribuição homogênea e não homogênea dos dados dentro das máquinas participantes.

1.1 Justificativa

Com base nos obstáculos que o trabalho com dados sensíveis pode trazer aos cientistas de dados, bem como a importância que os treinamentos podem ter para a evolução da ciência e a crescente popularidade do aprendizado federado, muitos especialistas enfrentam barreiras em termos de conhecimento. Isso ocorre porque o aprendizado federado também envolve a área de Sistemas Distribuídos. Portanto, é necessário criar ferramentas que facilitem o trabalho desses profissionais, permitindo que se concentrem apenas no aprendizado. Isso possibilita a criação e treinamento de modelos mais robustos e o uso de dados distribuídos.

1.2 Objetivos

Diante do exposto, o objetivo deste trabalho é apresentar um estudo referente ao aprendizado federado, com o intuito de simplificar e automatizar a implantação de sistemas distribuídos para esse fim. Para atingir o objetivo principal, as seguintes tarefas serão realizadas:

- Realizar uma revisão bibliográfica sobre *Federated Learning*;
- Realizar um estudo sobre a ferramenta a ser utilizada;
- Fazer a instalação da ferramenta em nuvem utilizando o serviço do Google cloud e analisar as diferença entre a implantação local e em nuvem;
- realizar alguns testes utilizando a base de dados *TissueMNIST* utilizando o *IBM FL* e um algoritmo de aprendizado local;
- Comparar o desempenho do algoritmo federado e o aprendizado tradicional;
- Criar notebooks para facilitar a utilização e instalação da ferramenta através de um tutorial;
- Realizar testes com os notebooks criados tanto em nuvem quanto localmente;
- Construção de um tutorial em vídeo utilizando as ferramentas criadas e que será disponibilizado no drive da universidade;

1.3 Organização do Trabalho

No Capítulo 2 serão apresentados os artigos da literatura sobre aprendizado federado, bem como as diversas ferramentas pesquisadas. Ademais, serão descritas as etapas de desenvolvimento do trabalho, juntamente com os procedimentos executados no Capítulo 3.

Por fim, serão apresentados os resultados obtidos, seguidos de reflexões acerca dos testes realizados no Capítulo 4. Para concluir, no Capítulo **??**, serão apresentados projetos a serem desenvolvidos futuramente, com o intuito de aprimorar e contribuir nos estudos acerca do aprendizado federado.

2 Revisão Bibliográfica

Neste capítulo, serão apresentados alguns artigos da literatura que abordam temáticas semelhantes ao tema deste trabalho, que se concentra na utilização das principais ferramentas disponíveis atualmente para o aprendizado federado. O estudo em questão contempla a utilização de ferramentas que permitem treinamentos de modelos de aprendizado de máquina em organizações que lidam com dados sensíveis, as quais não possuem configurações avançadas de hardware, acesso a serviços de internet de alta qualidade ou disponibilidade de recursos para o processamento de dados.

Assim, serão exploradas ferramentas que se mostram eficazes na resolução desses problemas e que possibilitam a realização de treinamentos em ambientes com limitações técnicas, sem comprometer a segurança dos dados sensíveis envolvidos.

2.1 Fundamentação Teórica

2.1.1 *Machine Learning*

De acordo com Carbonell, Michalski e Mitchell (1983), *Machine Learning* (ML) é um dos ramos existentes dentro da área de Inteligência Artificial (IA). O objetivo desse ramo é explorar a construção de algoritmos capazes de simular a capacidade humana de reconhecer padrões e identificar imagens, objetos ou qualquer tipo de dado, de forma autônoma, ou seja, sem interferência humana.

Atualmente, com o grande volume de dados que transita pela rede, que pode ultrapassar a casa dos terabytes, a análise desses dados é humanamente impossível. Por essa razão, o uso de *Machine Learning* tem crescido nos últimos anos, com o intuito de melhorar o desempenho e automatizar certas tarefas. Como exemplos de aplicações, podem ser citados a detecção de fraudes, resultados de pesquisa, previsão de falhas em equipamentos, além do reconhecimento de padrões em imagens, que é o foco deste trabalho.

2.1.2 *Federated Learning*

O aprendizado federado é uma técnica de aprendizado de máquina realizada de forma colaborativa, sem a centralização dos dados em um único local, como na abordagem tradicional. De acordo com o artigo de IBM (2020), o termo "*Federated Learning*" foi introduzido pelo Google em 2016, em um momento em que o uso inadequado de dados estava recebendo atenção da comunidade.

Em relação ao seu funcionamento (ver Figura 2.1), o Federated Learning utiliza um

modelo global para realizar o treinamento distribuído. À medida que os clientes participantes do treinamento executam um número pré-definido de épocas localmente, o modelo atualizado é criptografado e enviado para um agregador que fará a média com os demais modelos gerados, gerando um novo modelo global mais aprimorado.

Essa abordagem oferece várias vantagens, como a manutenção da privacidade dos dados, a disponibilização de mais materiais de amostra e o aumento do alcance da rede de pesquisa. Com isso, é possível proporcionar treinamentos mais efetivos para o Machine Learning e superar os desafios enfrentados pelos métodos tradicionais de aprendizado.

Esse ciclo se repete um número determinado de vezes, correspondente ao número de épocas globais. Por exemplo, um treinamento com dez épocas globais e três épocas locais terá um total de trinta épocas, com uma agregação do modelo aprimorado local a cada três épocas. Além disso, é possível definir a quantidade mínima de clientes por treinamento, a forma de agregação dos dados, os protocolos e a conexão utilizados.

Cabe ressaltar as diferenças entre o federated learning e o Distributed learning, visto que, nesta segunda abordagem, os dados de treinamento são distribuídos em várias máquinas e o modelo é treinado simultaneamente em todas as máquinas, usando os dados disponíveis em cada máquina. Os modelos locais são então enviados para um servidor central, onde são combinados para gerar um modelo global. Nessa abordagem, os dados são processados em diferentes máquinas, mas todos os dados são acessíveis a todas as máquinas, o que pode ser um problema de privacidade.

Em resumo, a principal diferença entre Federated Learning e Distributed Learning é a forma como os dados são distribuídos e processados, com o Federated Learning priorizando a privacidade do usuário ao manter os dados localmente e o Distributed Learning distribuindo dados em várias máquinas.

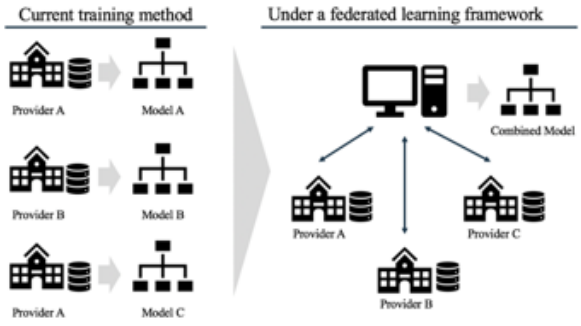

Figura 2.1 – Comparação entre treinamento centralizado e distribuído.
Fonte: <https://l1nq.com/gqJ9l>) [1]

2.1.3 PySyft

Durante o desenvolvimento deste trabalho, a primeira ferramenta testada foi a biblioteca PySyft, desenvolvida pela OpenMined. Essa ferramenta se destaca por seu uso de diversos

protocolos de segurança, como o *Secure Multi-Party Computation* (SMPC), *Homomorphic Encryption* (FHE/SHE) e *Functional Encryption* (FE), para possibilitar o aprendizado profundo e seguro (Théo Ryffel, 2022).

O PySyft é uma biblioteca de aprendizado federado que tem como base a biblioteca PyTorch, amplamente utilizada no campo de aprendizado de máquina. Por estar construído sobre essa biblioteca, sua implementação pode ser simplificada, tornando-se acessível aos usuários familiarizados com o PyTorch. No entanto, sua complexa arquitetura de segurança e criptografia pode apresentar dificuldades na agregação dos resultados obtidos, o que pode ser considerado como um desafio da ferramenta (Figura 2.2).

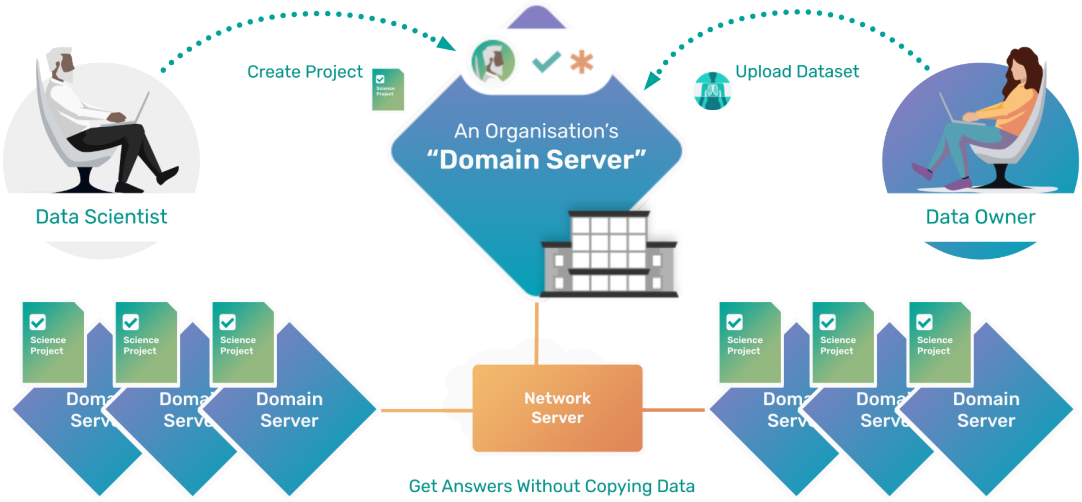

Figura 2.2 – Arquitetura PySyft.
Fonte: <https://github.com/OpenMined/PySyft>) [2]

2.1.4 IBM *Federated Learning Lib*

Durante o desenvolvimento deste trabalho, tivemos contato com a biblioteca de aprendizado federado desenvolvida pela IBM, que se apresentou como uma ferramenta de grande utilidade. A referida biblioteca oferece uma estrutura básica que pode ser facilmente personalizada para a execução de diversos modelos e bases de dados. Dessa forma, por não depender de uma estrutura específica, é possível realizar diversas configurações de topologia, agregação e protocolos. O IBM *Federated Learning* suporta redes neurais profundas (DNNs), regressão linear, k-means, além de abordagens supervisionadas e não supervisionadas (IBM, 2022a).

Um dos principais pontos positivos do IBM *Federated Learning* está relacionado com uma das dificuldades que os cientistas de dados enfrentam ao executar e configurar o aprendizado distribuído. A facilidade de uso da ferramenta é uma de suas maiores vantagens, permitindo a configuração rápida de diversos modelos e métodos de agregação já existentes na biblioteca, ou a criação de novos. Além disso, é possível realizar configurações para diversos ambientes, tais como data centers ou dispositivos de borda, como celulares.

No que diz respeito às funcionalidades compatíveis, a ferramenta da IBM suporta redes neurais de qualquer tipologia, desde que sejam suportadas pelas bibliotecas Keras, PyTorch e TensorFlow. Além das redes tradicionais, ela suporta Árvores de Decisão ID3, classificadores e regressões lineares, bem como algoritmos de *Deep Reinforcement Learning*.

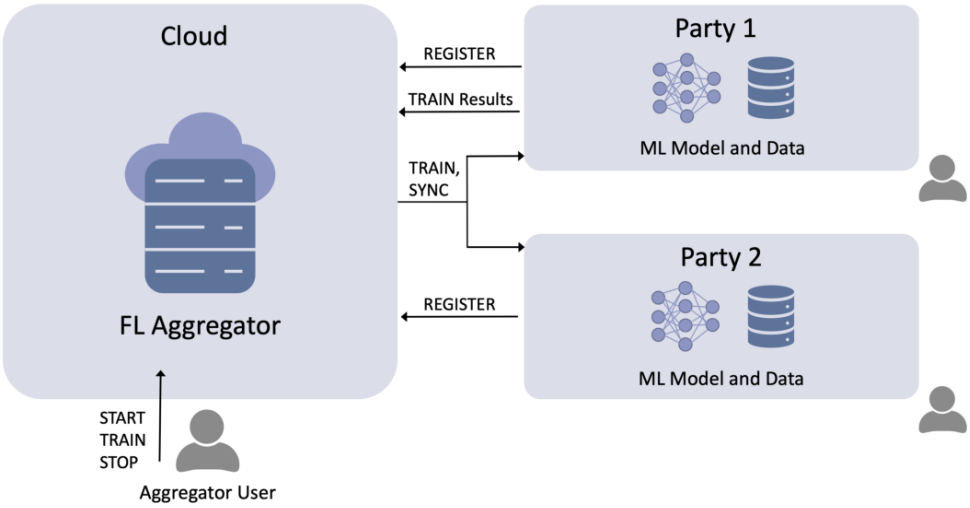

Figura 2.3 – Arquitetura IBM FL.
Fonte: Adaptação de IBM-FL. [3]

2.1.5 Serviço de Broker

Segundo artigo publicado por RedHat (2022), um broker é um serviço que promove a conexão entre consumidores e provedores de algum recurso. O broker detém informações sobre os serviços fornecidos e encaminha os pedidos solicitados pelo consumidor. Em outras palavras, o broker é responsável pelo intermédio entre o consumidor e o recurso desejado, automatizando processos que anteriormente eram realizados manualmente. É importante destacar que a utilização de brokers pode trazer inúmeros benefícios, tais como aumento da eficiência, segurança e redução de custos operacionais. Portanto, o estudo sobre brokers é fundamental para a compreensão e aprimoramento das práticas empresariais.

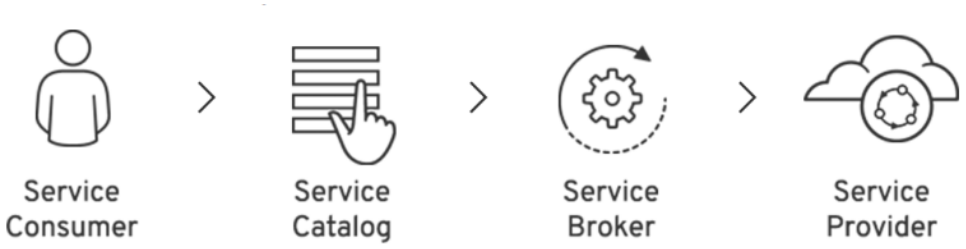

Figura 2.4 – Arquitetura Service Broker.
Fonte: Adaptação de IBM-FL. [4]

[3] Disponível em: <https://l1nq.com/J4qqY>. Acessado em 10 de outubro de 2022.
[4] Disponível em: <https://www.redhat.com/pt-br/topics/cloud-native-apps/service-brokers>. Acessado em 12 de outubro de 2022.

2.2 Trabalhos Relacionados

No trabalho realizado por Budrionis et al. (2021), para medir o desempenho do estudo federado, foi feita a comparação entre o modelo distribuído e a aprendizagem centralizada. Tais experimentos foram desenvolvidos sobre a base de dados MIMIC-III, que consiste em um conjunto de informações sobre pacientes que ficaram na UTI do Beth Israel Deaconess Medical Center entre os anos de 2001 a 2012, com o objetivo de classificar de forma binária a mortalidade intra-hospitalar. Segundo Lee e Shin (2020), o fato de o *Federated Learning* ser algo novo e, de certa forma, uma tecnologia imatura, coloca em xeque a confiabilidade, escalabilidade e precisão do mesmo. Contudo, em estudos realizados anteriormente, foi comprovado que, com apenas três nós, o aprendizado federado pode ser viável para o treinamento de modelos federados.

O estudo proposto por Budrionis et al. (2021)aborda a necessidade de realizar testes em diferentes configurações de rede, para que seja possível entender como a quantidade de nós e dados distribuídos podem afetar o desempenho do aprendizado. Os modelos foram avaliados em relação ao desempenho preditivo, duração do treinamento e inferência, além disso, a quantidade de dados e nós foram usados como parâmetros para medir a escalabilidade e desempenho do modelo. De acordo com o estudo, o treinamento realizado em nós virtuais e reais foi respectivamente 3 e 9 vezes mais lento em comparação com o aprendizado centralizado.

Algumas limitações de Budrionis et al. (2021)englobam a simplicidade do algoritmo usado, uma vez que o estudo pode não ser generalizável para um cenário de modelos de aprendizado mais avançados. Além disso, não foi considerada a distribuição das informações dentro dos nós, como exemplificado no texto: "um consultório pode atender mais crianças do que idosos". Diante disso, pode-se compreender que o desempenho do modelo pode ser afetado.

Dessa forma, baseado em Budrionis et al. (2021), é possível concluir que o aprendizado federado pode alcançar desempenho comparável ao aprendizado centralizado e que a distribuição não igualitária dos dados tende a não afetar o desempenho do treinamento. No entanto, treinar modelos de *Machine Learning* em ambientes federados aumenta o custo devido à complexidade de se orquestrar nós distribuídos, podendo demorar até 9 vezes mais que os modelos centralizados. Além disso, seu tempo de inferência pode aumentar em 40 vezes nos ambientes de computação em nuvem, em que os mesmos estão alocados em uma única rede, ou seja, a aplicação na vida real pode tornar esses números maiores.

Segundo Wei et al. (2019), com o surgimento dos aplicativos de Big Data, houve um aumento na preocupação e no cuidado com a segurança dos dados. Diante disso, podemos analisar que uma das vantagens do aprendizado de máquina tradicional é o treinamento local, que dificulta o acesso de invasores aos dados sensíveis. Entretanto, apesar de o treinamento federado ser realizado de forma privada, uma vez que os dados não se encontram nas mãos dos cientistas, as informações sensíveis ainda podem ser compartilhadas a partir da análise dos parâmetros e pesos da rede. Wei et al. (2019) propõem uma abordagem baseada na Privacidade Diferencial (PD) para

evitar de forma efetiva o vazamento de informações sensíveis. Tal abordagem adiciona ruídos aos dados transacionados pela rede, evitando que a captura de dados exponha alguma informação. No momento inicial, a partir de um parâmetro de criptografia global, os dados necessários para o treinamento são enviados aos clientes, que, por sua vez, executam o treinamento localmente e, no momento do envio, adicionam ruídos aos parâmetros, com base em critérios estabelecidos previamente. Em seguida, o servidor central, a partir do parâmetro global, agrega os valores recebidos.

De acordo com os resultados obtidos pelo estudo de Wei et al. (2019), os treinamentos que utilizaram a PD apresentaram uma relação inversamente proporcional entre o desempenho do treinamento e o nível de proteção da privacidade. Entretanto, experimentos com um número maior de clientes, mantendo um mesmo nível de privacidade, podem melhorar o desempenho do algoritmo.

No artigo de Hall et al. (2021), é apresentado o *framework* Syft na sua versão 0.5, cujo propósito geral é reforçar a privacidade dos dados no aprendizado federado. O trabalho inicia com uma reflexão sobre a complexidade da divulgação de dados pessoais ou imagens de pessoas, lembrando que, uma vez que os dados foram copiados, jamais podemos garantir que os mesmos foram destruídos. O artigo cita o caso do conjunto de dados DukeMTMC (RISTANI et al., 2016), que foi retirado devido ao uso não autorizado de imagens de rostos de pessoas para fins questionáveis, mas cópias derivadas ainda podem ser encontradas na internet, demonstrando a delicadeza da divulgação de dados pessoais.

O Syft oferece aos cientistas de dados a possibilidade de trabalhar com dados sensíveis de forma privada, uma vez que as variáveis remotas para o cientista são tratadas como ponteiros de rede para que possam ser operadas do lado do cliente. Além disso, para garantir uma maior privacidade e segurança das informações, depois de realizada uma conexão WebRTC (que permite comunicação ponto a ponto em tempo real e troca de dados), ainda é possível criptografar as informações usando uma abordagem segura de computação multipartidária (SyMPC) ou através do TenSEAL com a criptografia homomórfica.

Além dos métodos citados acima, os dados computados são mantidos temporariamente no cliente que realizou o treinamento, e cabe ao proprietário dos dados aprovar ou negar solicitações para acesso às informações geradas. O aceite pode ser realizado manualmente ou automaticamente por meio de políticas pré-determinadas.

Segundo Hall et al. (2021), uma das vantagens do Syft é o fato de a ferramenta possuir um design em formato de API, tornando o uso por parte do cientista de dados mais simples, uma vez que a existência de uma camada sobre seus componentes permite que códigos existentes sejam executados com alterações mínimas e a escrita de um novo código em Syft seja intuitiva. Dessa forma, espera-se que a complexidade de acesso aos métodos criptográficos da privacidade diferencial seja significativamente reduzida.

Em um trabalho desenvolvido pela IBM Ludwig et al. (2020), utilizando a própria ferramenta de aprendizado federado, são citadas algumas das adversidades do aprendizado federado retratadas em outros artigos, tais como a heterogeneidade dos dados, a robustez do processo e a seleção de operadores de fusão que sejam imparciais, entre outros. Segundo o estudo, a ferramenta IBM *Federated Learning* propõe uma estrutura para enfrentar as adversidades citadas e facilitar a integração do aprendizado federado de forma produtiva em empresas. Dessa forma, os profissionais podem se beneficiar de uma transição leve e eficiente das práticas tradicionais de escrita de aprendizado para modelos federados. O serviço de aprendizado federado deve ser de fácil implantação e se encaixar nos padrões de infraestrutura de TI, permitindo que qualquer cientista possa experimentar o aprendizado federado dentro de suas organizações sem alterações drásticas. O artigo apresenta os conceitos principais da ferramenta e, em seguida, a arquitetura do sistema, que consiste em uma biblioteca Python na qual é possível implementar um agregador responsável por coordenar o treinamento e uma parte que executa o treinamento. Segundo os autores, esse design permite uma estrutura que opere independentemente do algoritmo executado. A estrutura mencionada permite também que as partes pré-processem dados de diferentes localidades e formatos. Por fim, o estudo demonstra como realizar e configurar os treinamentos a partir de diferentes modelos, o que pode ser de grande ajuda, visto que durante a pesquisa, foi encontrada certa dificuldade em encontrar informações acerca da implantação e configuração das ferramentas testadas.

2.3 Seleção de ferramentas

O aprendizado federado tem se tornado cada vez mais relevante no campo de machine learning, especialmente em cenários em que a proteção da privacidade dos dados é uma preocupação primordial. Nesse sentido, a biblioteca PySyft se apresenta como uma alternativa viável para a implementação de soluções de aprendizado federado. No entanto, sua utilização não está isenta de desafios, tais como incompatibilidades com modelos em PyTorch e Keras em algumas versões, dificuldades de conexão com grande quantidade de máquinas e consumo excessivo de recursos da máquina. Esta seção descreve as etapas enfrentadas durante a implantação da biblioteca PySyft, conforme ilustrado no fluxograma apresentado na Figura 2.5, a fim de superar os desafios mencionados e alcançar uma solução satisfatória.

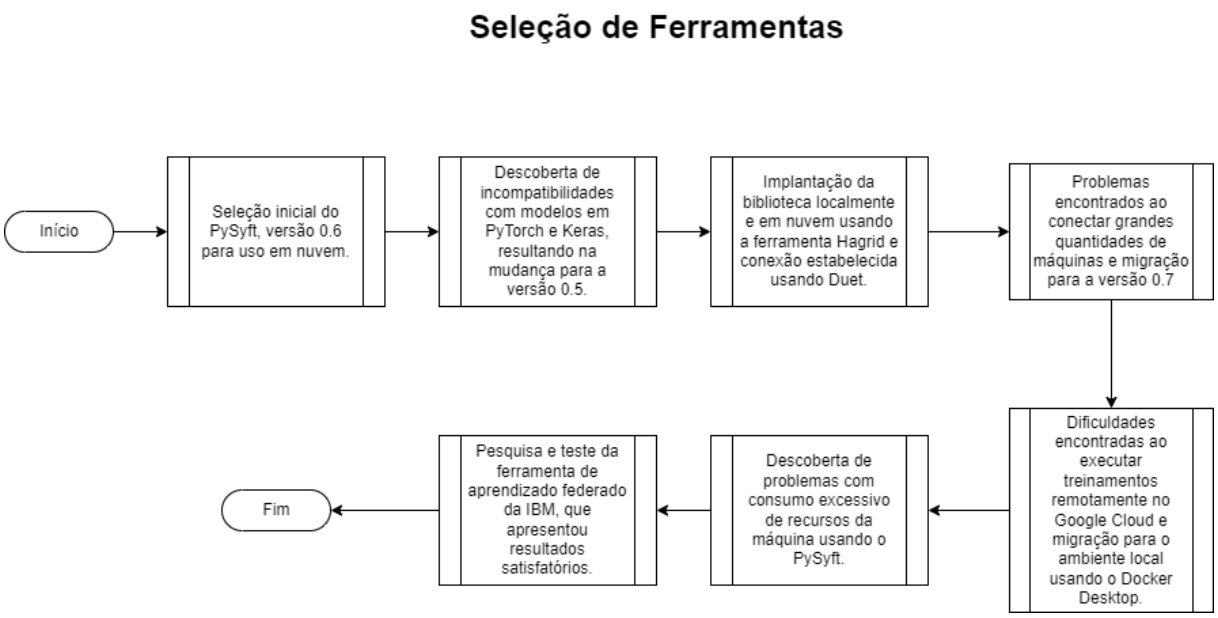

Figura 2.5 – Fluxograma da seleção de ferramentas.
Fonte: Elaborado pelo autor.

A primeira ferramenta utilizada neste trabalho foi o PySyft, uma biblioteca Python para privacidade e segurança em aprendizado de máquina distribuído. Inicialmente, a versão 0.6 do PySyft foi implantada no Google Cloud, pois é a versão mais estável. No entanto, não foi encontrada documentação explícita sobre a compatibilidade da biblioteca com a utilização de modelos em PyTorch ou Keras. Embora a própria desenvolvedora da ferramenta tenha informado que esses modelos podem ser utilizados.

Ao aprofundarmos nossa pesquisa e entrarmos em contato com os desenvolvedores da ferramenta por meio do canal oficial do Slack, descobrimos que, por estar em fase de desenvolvimento e por estarem testando novas funcionalidades, os desenvolvedores não garantem retrocompatibilidade entre as versões. Por esse motivo, foi possível encontrar exemplos de redes neurais para treinamento de Machine Learning nas versões anteriores, mas não na versão 0.6.

A partir desse momento, optamos por migrar para a versão 0.5 do PySyft, com a qual foi possível implantar a biblioteca tanto localmente quanto em nuvem. A ferramenta possui uma interface de linha de comando (CLI) para implantação, chamada Hagrid, que faz o deploy de um contêiner Docker com todas as imagens necessárias para a aplicação, como pode ser visto na figura 2.6, a partir do comando apresentado no Código 2.1.

```
1 hagrid launch <node name> domain to localhost
```

Código 2.1 – Linha de execução com Hagrid.

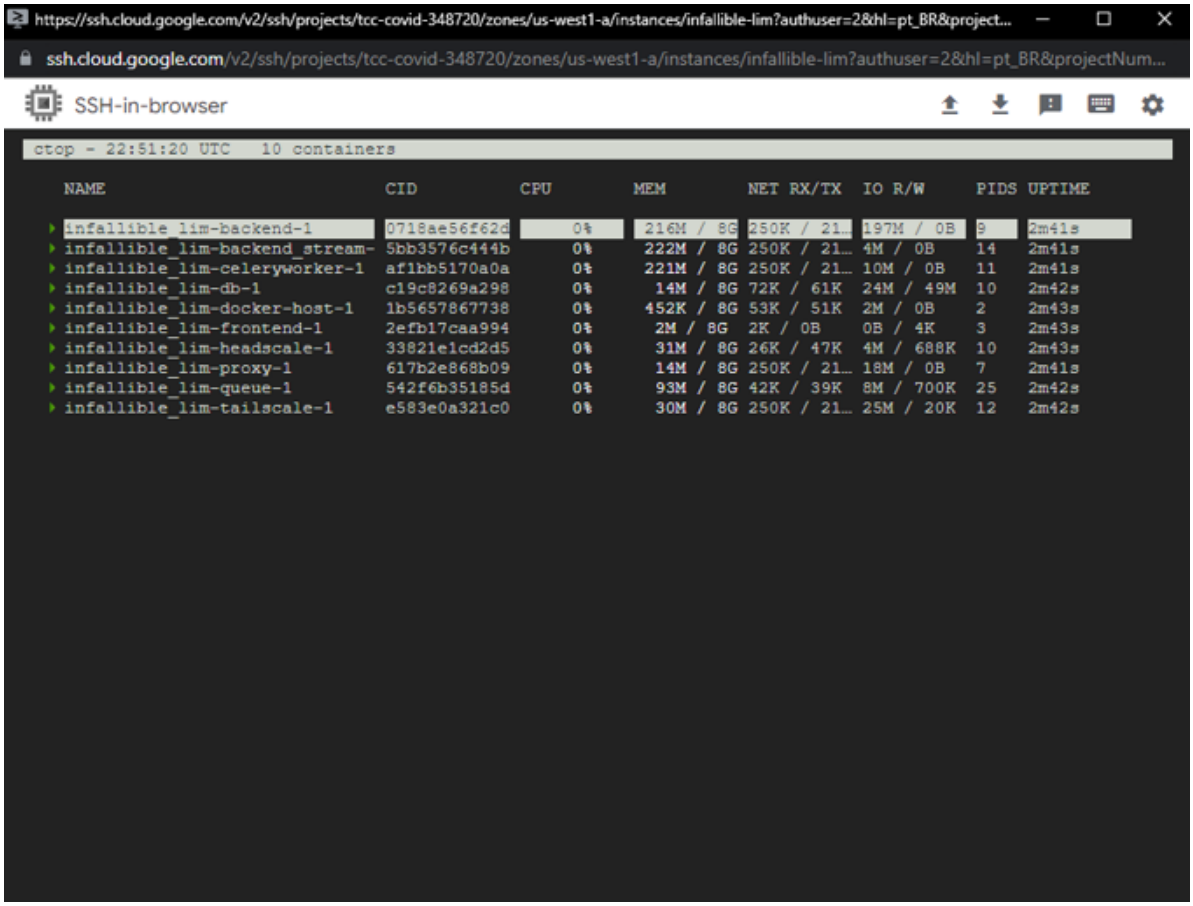

Figura 2.6 – *Containners* do *deploy* no *Google Cloud*.
Fonte: Elaborado pelo autor.

Segundo documentação oficial do PySyft (2021), a versão 0.5 utiliza uma ferramenta chamada Duet para estabelecer a conexão entre a máquina que realizará o treinamento e o cientista de dados. A conexão entre as duas sessões é estabelecida por meio de uma conexão peer-to-peer utilizando WebRTC e STUN, que são protocolos de conexão em tempo real. Como essa é uma conexão ponto a ponto, é necessário que o proprietário dos dados gere um código para que o cientista de dados se conecte à rede e, em seguida, um código é gerado para que o cientista de dados envie ao proprietário e concretize o vínculo.

Uma desvantagem dessa forma de conexão é que, caso seja necessário realizar um treinamento com um grande número de clientes, torna-se inviável conectar manualmente todas as máquinas. Tendo em vista que o objetivo deste trabalho é facilitar a utilização por parte dos cientistas de dados, novas formas de executar o treinamento foram exploradas.

Para solucionar o problema de ter que se conectar manualmente a cada um dos clientes, foi migrado para a versão 0.7, que ainda está em desenvolvimento. Ao contrário da versão 0.6, a versão 0.7 apresenta exemplos e compatibilidade com modelos em Pytorch e Keras. Além disso, assim como na versão 0.6, essa versão não utiliza o Duet e, por isso, é possível conectar-se diretamente com os clientes apenas com o IP da máquina e as credenciais do cientista de dados, como na figura 2.7.

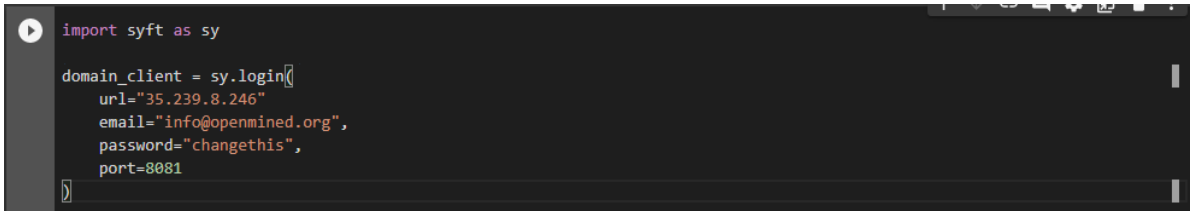

Figura 2.7 – Login utilizando IP externo do *Google Cloud*.

Para realizar os treinamentos, utilizamos a base de dados anteriormente descrita e a rede de exemplo disponível em um dos notebooks presentes no repositório de PySyft da biblioteca (GITHUB, 2022), conforme exemplificado no Código 2.2. Contudo, ao tentar executá-los remotamente pelo Google Cloud, encontramos dificuldades em realizar o upload dos dados e executar o treinamento. Notamos um tempo de execução muito alto e constantes perdas de conexão com a máquina.

Decidimos então migrar para um ambiente local e executar os mesmos procedimentos por meio da ferramenta Docker Desktop. Nesse momento, observamos um uso excessivo de CPU e memória RAM da máquina, que chegou perto dos cem por cento. Buscando soluções na comunidade do Slack, descobrimos que não éramos os únicos a enfrentar essa situação. Entretanto, não foi encontrada nenhuma solução ou explicação para o motivo desse consumo excessivo de recursos da máquina.

```
from syft.core.tensor import nn

model_net = nn.Model()
model_net.add(nn.Convolution(1, (3, 3), input_shape=(None, 1, 28, 28)))
model_net.add(nn.MaxPool((2, 2)))
model_net.add(nn.Convolution(2, (4, 4)))
model_net.add(nn.MaxPool((2, 2)))
User call .compile to initialize model. This sends the model to domain
    and returns a model pointer.
model_ptr = model_net.compile(to=domain, loss=BinaryCrossEntropy(),
    optimizer=Adamax())
User calls .fit method on the model pointer and passes image_ptr and
    label_ptr as inputs

model_ptr.fit(X_train_ptr, y_train_ptr, max_iter=10, validation_split
    =0.1, batch_size=100)
Monitor training progress. This prints information like {"loss": "", "
    epoch": "", .....}

model_ptr.progress()
User call .weights methods to get weights pointer
model_weights_ptr = model_ptr.weights()
published_weights = model_weights.publish()
```

```
19 public_weights = published_weights.get()
```

Código 2.2 – Código de exemplo disponível no GitHub do *PySyft*

Por fim, com o objetivo de encontrar uma ferramenta que apresentasse um bom desempenho e fosse mais eficiente em comparação as demais ferramentas, como exemplificado em 2.1, foram realizadas pesquisas e testes de instalação e execução utilizando a ferramenta de aprendizado federado desenvolvida pela IBM. Os resultados obtidos foram satisfatórios (IBM, 2022b). Nas próximas seções, serão descritos os procedimentos realizados para a realização dos testes e análise dos resultados.

Ferramenta	Versão	Pontos Positivos	Pontos Negativos
PySyft	0.5	- Facilidade de uso para usuários iniciantes - Boa documentação - Suporte para múltiplas linguagens de programação	- Limitações na implementação de algoritmos de aprendizado federado
PySyft	0.6	- Melhorias na implementação de algoritmos de aprendizado federado - Integração com outras bibliotecas de aprendizado de máquina	- Necessidade de conhecimento prévio em programação distribuída
PySyft	0.7	- Implementação de novos recursos para segurança e privacidade - Suporte para aprendizado federado com dados não i.i.d.	- Performance limitada em cenários de aprendizado federado em larga escala
IBM FL	1.0.7	- Integração com outras ferramentas IBM para análise de dados e gerenciamento de modelos - Escalabilidade - Recursos de segurança e privacidade avançados	- Número maior de passos para instalação e configuração

Tabela 2.1 – Comparação entre as versões do PySyft e do IBM FL

3 Desenvolvimento

Neste capítulo será abordado todos os instrumentos e procedimentos metodológicos, apresentados ou não na fundamentação teórica, que serão utilizados para alcançar o objetivo proposto.

3.1 Base de dados

A primeira base de dados utilizada foi a *PneumoniaMNIST*, a qual contém imagens de raio-x de pulmões com pneumonia. O conjunto consiste em uma classificação binária, composto por 4.708 imagens de treinamento, 524 imagens de validação e 624 imagens de teste. O objetivo ao utilizar este conjunto de imagens foi realizar testes rápidos, já que o número de imagens é relativamente baixo em comparação com outras bases do conjunto, o que resultaria em um tempo de execução menor. Entretanto, durante os testes, o modelo convergiu rapidamente e estagnou após apenas três rodadas de treinamento, o que impossibilitou a configuração dos experimentos.

A base *TissueMNIST* (Figura 3.1) faz parte do conjunto de dados médicos denominado *MedMNIST* (YANG et al., 2021), que consiste em imagens biomédicas padronizadas, semelhantes ao conjunto de dados MNIST. O *MedMNIST* contém doze conjuntos de dados 2D e seis conjuntos 3D, conforme apresentado na Tabela 3.1. Assim como no conjunto de dados MNIST, as imagens foram pré-processadas em 28x28 (2D) e 28x28x28 (3D), e uma das vantagens deste conjunto é que ele já possui as *labels* de classificação, o que não requer nenhum conhecimento prévio para realizar os treinamentos.

Figura 3.1 – Exemplos da base TissueMNIST.
Fonte: (YANG et al., 2021).

Algumas das características do *MedMNIST* consistem em sua diversificação, que inclui uma grande variedade de imagens de doenças catalogadas, classificações binárias, multiclasse,

multi-rótulos e regressão ordinal. Além disso, todos os subconjuntos de dados são pré-processados e possuem divisões de treinamento, teste e validação padronizadas.

A base *TissueMNIST* consiste em um conjunto de imagens de células do córtex do rim humano coletadas de três amostras de tecido e organizadas em oito categorias diferentes. O conjunto é composto por 165 mil imagens de treinamento, 23 mil imagens de validação e 47 mil imagens de teste, todas em escala de cinza e no formato 28x28.

Nome	Dados	Classe	Treino/Validação/Teste
PathMNIST	Patologia do cólon	Multiclasse (9)	89,996 / 10,004 / 7,180
ChestMNIST	Raio-X de tórax	Multi-Rótulo (14) Classe Binária (2)	78,468 / 11,219 / 22,433
DermaMNIST	Dermatoscópio	Multiclasse (7)	7,007 / 1,003 / 2,005
OCTMNIST	OCT da retina	Multiclasse (4)	97,477 / 10,832 / 1,000
PneumoniaMNIST	Raio-X de tórax	Classe Binária (2)	4,708 / 524 / 624
RetinaMNIST	Câmera do fundo do olho	Regressão Ordinal (5)	1,080 / 120 / 400
BreastMNIST	Ultrassom de mama	Classe Binária (2)	546 / 78 / 156
BloodMNIST	Microscópio de células sanguíneas	Multiclasse (8)	11,959 / 1,712 / 3,421
TissueMNIST	Microscópio de córtex renal	Multiclasse (8)	165,466 / 23,640 / 47,280
OrganAMNIST	TC abdominal	Multiclasse (11)	34,581 / 6,491 / 17,778
OrganCMNIST	TC abdominal	Multiclasse (11)	13,000 / 2,392 / 8,268
OrganSMNIST	TC abdominal	Multiclasse (11)	13,940 / 2,452 / 8,829

Tabela 3.1 – Sumário da base *MedMNIST*.
Fonte: Adaptado de (YANG et al., 2021).

3.2 Métodos

Nessa seção, serão explicitados os procedimentos e escolhas feitas para a execução das ferramentas necessárias para o aprendizado federado.

3.2.1 Aplicando o aprendizado federado com a ferramenta da IBM

Nesta seção, serão descritos os procedimentos adotados para a implantação da ferramenta selecionada tanto em ambiente local quanto em nuvem, conforme ilustrado na figura 3.2. Além disso, serão apresentados os critérios utilizados na escolha do modelo, a seleção dos parâmetros e as configurações adotadas.

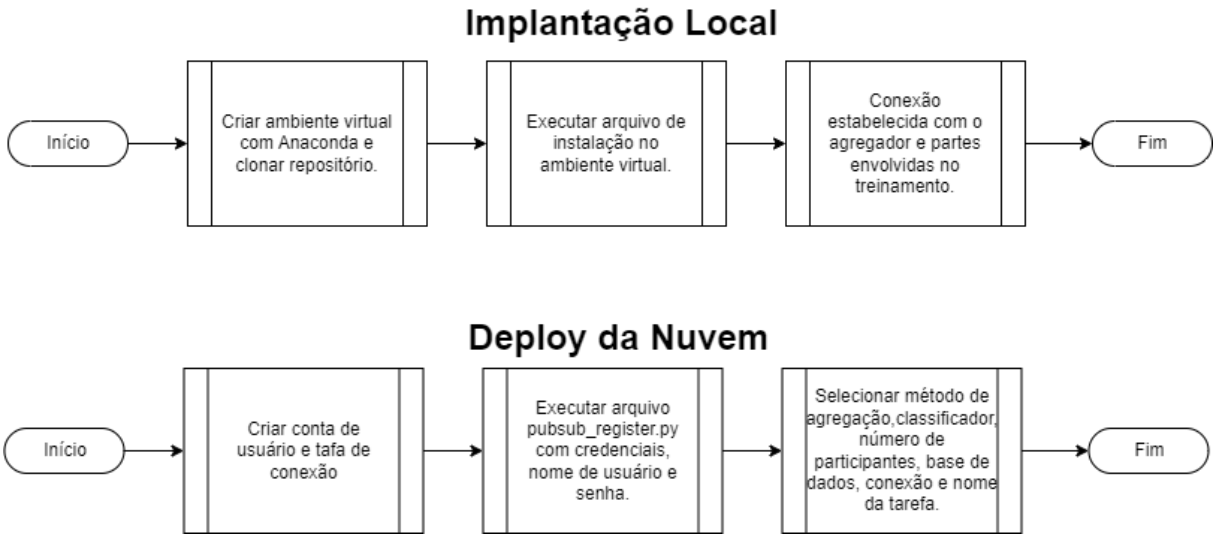

Figura 3.2 – Fluxograma de implantação da ferramenta.
Fonte: Elaborado pelo autor.

3.2.1.1 Implantação local

Para implantação da ferramenta disponibilizada pela IBM é preciso criar um ambiente virtual, para realização desse trabalho foi utilizado o programa Anaconda, seguindo os comandos do código presente em 3.1. Após a ativação do ambiente é preciso clonar seu repositório no GitHub. Em seguida, para realizar a instalação é necessário executar o arquivo presente no repositório em questão.

```
conda create -n <env_name> python=3.6 tensorflow=1.15
conda activate <env_name>
git clone https://github.com/IBM/federated-learning-lib.git
cd .\federated-learning-lib\
cd .\federated-learning-lib\
pip install <IBM_federated_learning_whl_file>
```

Código 3.1 – Criação do ambiente de execução

3.2.1.2 Deploy da nuvem

A IBM disponibiliza um *plugin PubSub* para utilizar a ferramenta em nuvem. Este *plugin* utiliza um serviço de *broker* que consiste em uma instância do *RabbitMQ* implantada no *IBM Cloud*. O objetivo deste sistema é garantir uma conexão segura entre o agregador e as partes que executam o treinamento (IBM, 2022c).

Considerando que o *broker* está em execução no *IBM Cloud*, é necessário criar uma conta de usuário para todas as partes envolvidas no treinamento e uma tarefa para executá-lo, conforme exemplificado no Código 3.2. Para criar os registros, há um arquivo Python disponível na ferramenta, denominado "`pubsub_register.py`", que requer como parâmetros as credenciais

presentes na pasta do projeto, um "aggregator user- nome de usuário do agregador - e uma senha. O mesmo procedimento aplica-se às partes que executarão o treinamento.

Após a criação dos registros para cada agente envolvido no treinamento, é preciso criar uma tarefa que execute a conexão. Para isso, são necessárias apenas as credenciais e um nome. Em seguida, para finalizar a configuração do *broker*, é necessário selecionar o método de agregação - no exemplo foi escolhido "iter_avg-, o classificador, que foi especificado com Keras, a quantidade de participantes, a base de dados a ser trabalhada e sua localização, o tipo de conexão que estamos configurando e, por fim, o nome da tarefa criada.

```
python examples/pubsub_register.py
    --credentials=pubsub_credentials.json
    --user=<AGGREGATOR USER>
    --password=<PASSWORD> > aggregator.json

python examples/pubsub_register.py
    --credentials=pubsub_credentials.json
    --user=<PARTY 0>
    --password=<PASSWORD> > party0.json

python examples/pubsub_register.py
    --credentials=pubsub_credentials.json
    --user=<PARTY N>
    --password=<PASSWORD> > partyn.json

python examples/pubsub_task.py
    --credentials=aggregator.json
    --task_name=<TASK NAME>

python examples/generate_configs.py -f iter_avg
                                    -m keras
                                    -n 2
                                    -d mnist
                                    -p examples/data/mnist/random
                                    -c pubsub
                                    -t <TASK NAME>
```

Código 3.2 – Comandos para configuração do *pubsub*

3.2.1.3 Modelo de *Deep Learning*

O modelo utilizado na fase experimental da ferramenta possui uma arquitetura simples, exemplificada no Código 3.3. Nessa fase, o objetivo do trabalho consiste em configurar e executar alguns exemplos para compreender o funcionamento do aprendizado federado.

```
import os

```

```
import keras
from keras import backend as K
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Dropout, Flatten
from keras.models import Sequential
def save_model_config(folder_configs):
    num_classes = 10
    img_rows, img_cols = 28, 28
    if K.image_data_format() == 'channels_first':
        input_shape = (1, img_rows, img_cols)
    else:
        input_shape = (img_rows, img_cols, 1)

    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=input_shape))
    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='softmax'))

    model.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adadelta(),
                  metrics=['accuracy'])

    if not os.path.exists(folder_configs):
        os.makedirs(folder_configs)

    # Save model
    fname = os.path.join(folder_configs, 'compiled_keras.h5')
    model.save(fname)
    print(fname)

    K.clear_session()
    # Generate model spec:
    spec = {
        'model_name': 'keras-cnn',
        'model_definition': fname
    }

    model = {
        'name': 'KerasFLModel',
        'path': 'ibmfl.model.keras_fl_model',
```

```
50          'spec': spec
51      }
52
53      return model
54  save_model_config('examples/configs/keras_classifier')
```

Código 3.3 – Modelo criado com TensorFlow/Keras.

A arquitetura do modelo é composta por duas camadas de convolução, sendo a primeira contendo 32 filtros diferentes e a segunda com 64. Ambas possuem ativação do tipo ReLU, que transforma valores menores que zero em zero e mantém os valores maiores que esse limiar. Em seguida, é aplicado o MaxPooling e o Dropout, onde o primeiro reduz o tamanho dos filtros pela metade, e o segundo descarta 25% das conexões. Essas ações visam fortalecer a rede e torná-la mais consistente.

Por fim, é adicionado o Flatten, que transforma o volume gerado pelas convoluções e poolings em um vetor para que possa ser utilizado pela camada Dense com 128 neurônios, a qual se conecta com os neurônios resultantes do Flatten.

Na última linha do modelo, é utilizado o comando "compile", que recebe como parâmetros um otimizador e uma função de perda. No exemplo em questão, foi escolhido o otimizador "adam", que é baseado na descida de gradiente, e a função de perda "categorical_crossentropy", utilizada para quantificar a perda de modelos multiclasse com duas ou mais saídas. Por fim, é definido um batch size de 128.

3.2.1.4 Treinamento da rede no ambiente federado

Nas seções anteriores, foi descrita a forma de implantação da ferramenta da IBM. Contudo, após a implantação, é necessário gerar os arquivos de configuração do treinamento. Conforme exemplificado no Código 3.4, já existem arquivos base de geração de configurações dentro da ferramenta. Para gerar as configurações para nosso treinamento, especificamos o algoritmo *FedAvg* utilizando a biblioteca Keras. Serão utilizadas duas máquinas para executar o treinamento, enquanto uma terceira funcionará como agregadora dos modelos gerados. Por fim, faremos uso de uma base de dados customizada.

```
1  python examples/generate_configs.py -f fed_avg
2                                      -m keras
3                                      -n 2
4                                      -d custom_data
```

Código 3.4 – Gerar configuração de treinamento

Após a geração do arquivo de configuração, teremos um arquivo `.yml` contendo as especificações do agregador de treinamento presente no Código 3.5 e das partes executantes nos Códigos 3.7 e 3.8.

O agregador carrega consigo informações de conexão referentes à máquina de execução em que está instalado. Ademais, o arquivo de configuração contém informações relevantes ao algoritmo de fusão, hiperparâmetros do treinamento e protocolo a serem utilizados.

```
connection:
  info:
    ip: 127.0.0.1
    port: 5000
    tls_config:
      enable: false
  name: FlaskConnection
  path: ibmfl.connection.flask_connection
  sync: false
fusion:
  name: FedAvgFusionHandler
  path: ibmfl.aggregator.fusion.fedavg_fusion_handler
hyperparams:
  global:
    max_timeout: 600000
    num_parties: 2
    rounds: 3
    termination_accuracy: 0.9
  local:
    training:
      epochs: 3
protocol_handler:
  name: ProtoHandler
  path: ibmfl.aggregator.protohandler.proto_handler
```

Código 3.5 – Configurações do agregador

Conforme mencionado anteriormente nesta seção, a base de dados utilizada neste trabalho não está disponível nos exemplos da biblioteca. Portanto, é necessário criar um manipulador de leitura de arquivos. O código responsável por essa tarefa pode ser encontrado no Código 3.6, e o nome e o local do arquivo devem ser especificados no arquivo de configuração das máquinas que foram utilizadas para realizar o treinamento.

```
def load_dataset(self, nb_points=500):
    try:
        data_train = np.load(self.file_name)
        x_train = data_train['train_images']
        y_train = data_train['train_labels']
        x_test = data_train['test_images']
        y_test = data_train['test_labels']
    except Exception:
        raise IOError('Unable to load training')
    return (x_train, y_train), (x_test, y_test)

def preprocess(self):
    num_classes = 8
    img_rows, img_cols = 28, 28
```

```
15
16      if self.channels_first:
17          self.x_train = self.x_train.reshape(self.x_train.shape[0], 1, img_rows, img_cols)
18          self.x_test = self.x_test.reshape(self.x_test.shape[0], 1, img_rows, img_cols)
19      else:
20          self.x_train = self.x_train.reshape(self.x_train.shape[0], img_rows, img_cols, 1)
21          self.x_test = self.x_test.reshape(self.x_test.shape[0], img_rows, img_cols, 1)
22
23      self.y_train = np.eye(num_classes)[self.y_train]
24      self.y_test = np.eye(num_classes)[self.y_test]
25      self.y_train = self.y_train.reshape((-1,num_classes))
26      self.y_test = self.y_test.reshape((-1,num_classes))
```

Código 3.6 – *Data handler* criado para a base de dados *MedMNIST*.

O *handler* funciona como um interpretador da base de dados TissueMNIST, permitindo a separação dos dados em conjuntos de treino, validação e teste. Ademais, realiza-se o pré-processamento dos dados para que estes estejam no formato esperado pelo modelo apresentado na Seção 3.2.1.3.

Juntamente com as configurações do agregador, são geradas configurações específicas para cada máquina participante da execução do treinamento, de modo que cada máquina possui um arquivo de configuração próprio. Este arquivo, assim como o do agregador, contém informações de conexão específicas.

Outra informação fundamental para a execução do treinamento é o tipo de conexão utilizado, que, conforme mostrado no Código 3.7, utiliza *flask*. Além disso, informações sobre a base de dados, especificidades do treinamento local, modelo, informações de privacidade e protocolo são necessárias para a configuração do treinamento.

Ressalta-se que a base de dados foi dividida em duas partes iguais, cada uma contendo 50% do total de imagens. Dessa forma, cada parte é referência para uma das metades da base de dados. No exemplo presente no Código 3.8, é possível notar que o nome do arquivo utilizado para o treinamento difere do exemplo apresentado no Código 3.7.

```
1  aggregator:
2    ip: 127.0.0.1
3    port: 5000
4  connection:
5    info:
6      ip: 127.0.0.1
7      port: 8085
8      tls_config:
9        enable: false
10   name: FlaskConnection
11   path: ibmfl.connection.flask_connection
12   sync: false
13 data:
14   info:
15     npz_file: examples/data/tissuemnist/random/data_party0.npz
```

```
  name: PneumoniaMedmnistDataHandler
  path: examples/configs/iter_avg/pytorch/medHandler.py
local_training:
  name: FedAvgLocalTrainingHandler
  path: ibmfl.party.training.fedavg_local_training_handler
model:
  name: KerasFLModel
  path: ibmfl.model.keras_fl_model
  spec:
    model_definition: examples/configs/fedavg/keras/compiled_keras.h5
    model_name: keras-cnn
privacy:
  metrics: true
protocol_handler:
  name: PartyProtocolHandler
  path: ibmfl.party.party_protocol_handler
```

Código 3.7 – Configurações da parte 0

```
aggregator:
  ip: 127.0.0.1
  port: 5000
connection:
  info:
    ip: 127.0.0.1
    port: 8086
    tls_config:
      enable: false
  name: FlaskConnection
  path: ibmfl.connection.flask_connection
  sync: false
data:
  info:
    npz_file: examples/data/tissuemnist/random/data_party1.npz
  name: PneumoniaMedmnistDataHandler
  path: examples/configs/iter_avg/pytorch/medHandler.py
local_training:
  name: FedAvgLocalTrainingHandler
  path: ibmfl.party.training.fedavg_local_training_handler
model:
  name: KerasFLModel
  path: ibmfl.model.keras_fl_model
  spec:
    model_definition: examples/configs/fedavg/keras/compiled_keras.h5
    model_name: keras-cnn
privacy:
  metrics: true
protocol_handler:
```

```
30  name: PartyProtocolHandler
31  path: ibmfl.party.party_protocol_handler
```

Código 3.8 – Configurações da parte 1

Para realizar os treinamentos neste trabalho, foi utilizado o algoritmo de aprendizado federado chamado Federated Averaging (FedAvg) (SUN; LI; WANG, 2021). O FedAvg envolve várias atualizações de um algoritmo chamado stochastic gradient descent (SGD) para encontrar os melhores parâmetros do modelo que aproximem as saídas previstas e reais. Com o FedAvg, é possível executar diversas épocas locais e, somente ao final, enviar ao servidor central apenas os pesos atualizados do treinamento, permitindo a agregação global de todos os parâmetros calculados localmente.

Dentro da biblioteca de aprendizado federado utilizada, existem comandos específicos necessários para a execução do treinamento. Conforme apresentado na Figura 3.3, nem todos os comandos podem ser executados por todos os agentes. Para iniciar cada uma das partes e o agregador, é necessário executar o comando `START` para iniciar o servidor flask. Em seguida, as partes devem se registrar no agregador, presente em seu arquivo de configuração, utilizando o comando `REGISTER`. Por fim, o agregador pode iniciar o treinamento executando o comando `TRAIN`.

FL Command	Participant	Description
`START`	aggregator / party	Start accepting connections
`REGISTER`	party	Join an FL project
`TRAIN`	aggregator	Initiate training process
`SYNC`	aggregator	Synchronize model among parties
`STOP` (coming soon)	aggregator	Pause training process
`EVAL`	party	Evaluate model

Figura 3.3 – Comandos de terminal da biblioteca IBM *Federated Learning*.
Fonte: <https://github.com/IBM/federated-learning-lib/blob/main/setup.md>

Ao final de cada treinamento, é possível que o agregador sincronize o modelo resultante com todos os participantes por meio do comando `SYNC`. Dessa forma, uma última agregação é realizada e o modelo é enviado novamente para cada participante. Por fim, é possível solicitar a avaliação do modelo de acordo com as métricas apresentadas na Figura 3.4.

```
23640/23640 [==============================] - 11s 477us/step
2022-10-08 11:26:35,022 | 1.0.6 | INFO | ibmfl.model.keras_fl_model                  | {'f1 micro': 0.54, 'precision m
icro': 0.73, 'recall micro': 0.43, 'f1 macro': 0.36, 'precision macro': 0.65, 'recall macro': 0.28, 'f1 weighted': 0.51,
 'precision weighted': 0.71, 'recall weighted': 0.43}
2022-10-08 11:26:35,022 | 1.0.6 | INFO | ibmfl.model.keras_fl_model                  | {'loss': 0.23102561384691603, '
acc': 0.9094860405284539, 'f1 micro': 0.54, 'precision micro': 0.73, 'recall micro': 0.43, 'f1 macro': 0.36, 'precision
macro': 0.65, 'recall macro': 0.28, 'f1 weighted': 0.51, 'precision weighted': 0.71, 'recall weighted': 0.43}
2022-10-08 11:26:35,023 | 1.0.6 | INFO | ibmfl.party.training.local_training_handler   | {'loss': 0.23102561384691603, '
acc': 0.9094860405284539, 'f1 micro': 0.54, 'precision micro': 0.73, 'recall micro': 0.43, 'f1 macro': 0.36, 'precision
macro': 0.65, 'recall macro': 0.28, 'f1 weighted': 0.51, 'precision weighted': 0.71, 'recall weighted': 0.43}
2022-10-08 11:26:35,026 | 1.0.6 | INFO | werkzeug                                   | 127.0.0.1 - - [08/Oct/2022 11:2
```

Figura 3.4 – Resultado do comando de avaliação de um treinamento.
Fonte: Elaborado pelo autor

3.3 Tutorial

Nesta seção, serão explicitados os procedimentos e escolhas realizados para a criação de um tutorial que facilite a implantação e utilização do aprendizado federado mencionado na Seção 3.2. Durante o desenvolvimento do tutorial, houveram alguns obstáculos ao tentar automatizar todo o processo, desde a instalação até a execução dos treinamentos. Versões específicas de algumas bibliotecas tiveram que ser instaladas, portas tiveram que ser abertas nas máquinas virtuais e configurações da ferramenta Jupyter Notebook tiveram que ser realizadas para que o tutorial fosse executado tanto localmente quanto em máquinas virtuais.

3.3.1 Seleção da ferramenta para confecção dos tutoriais

Para a elaboração dos tutoriais, foi escolhida a ferramenta Jupyter Notebook, que se apresenta como uma ferramenta altamente útil para cientistas de dados. Isso se deve ao fato de que ela permite ao usuário criar e compartilhar documentos que contêm código ativo, equações, visualizações e narrativas em texto. Ademais, essa ferramenta pode ser empregada tanto para a limpeza, transformação e modelagem de dados, quanto para o desenvolvimento de algoritmos e modelos de machine learning. Dessa maneira, ela oferece um ambiente interativo para que os usuários possam explorar e investigar seus dados. Outro aspecto relevante do Jupyter Notebook é a possibilidade de integração com outras linguagens de programação, como Python, R, dentre outras, o que expande significativamente as possibilidades de uso e aplicação da ferramenta.

Ao final do processo de desenvolvimento dos Notebooks, foram obtidos três arquivos: um voltado para a preparação do ambiente, um para a execução do agregador e outro para a execução dos participantes. A partir dos notebooks, o cientista encontrará todos os procedimentos necessários para a execução do treinamento. Ademais, será possível alterar facilmente as células, de modo que o tutorial se adapte aos dados e modelos trabalhados. É importante salientar que essa adaptação deve ser realizada com cautela e seguindo as boas práticas da área em questão. Para garantir a reprodutibilidade dos resultados, é recomendável que sejam registradas todas as modificações realizadas nos Jupyter Notebooks e que sejam seguidos os padrões de nomenclatura e organização adotados pela equipe responsável pelo projeto.

No processo de criação dos tutoriais, utilizaram-se máquinas virtuais disponíveis no plano gratuito e ilimitado da plataforma Oracle Cloud. Foram criadas duas instâncias de Computação baseada em AMD, cada uma equipada com núcleos Ampere A1 baseados em ARM, contendo 1/8 de CPU e 1 GB de memória.

Devido às limitações do plano gratuito da Oracle Cloud, foram criadas duas máquinas virtuais adicionais no serviço da Amazon Cloud. Essas instâncias possuem SSD, ao contrário das máquinas da Oracle, que possuem HD, e possuem 1 CPU e 1 GB de memória cada, além de possuir um processador i3 de arquitetura x86. Com isso, foram utilizadas um total de quatro máquinas para o desenvolvimento dos tutoriais.

3.3.2 Criação do Notebook para preparação do ambiente

Para a instalação da ferramenta de aprendizado federado, é necessário o uso da versão 3.6 do Python e a versão 1.15 do TensorFlow. A fim de evitar conflitos com possíveis versões já instaladas nas máquinas utilizadas para treinamento, optou-se pela utilização da plataforma Anaconda. Essa ferramenta permite a instalação de diferentes versões do Python e suas dependências de forma isolada.

Após a instalação do Anaconda, é necessário a criação de um ambiente virtual a partir dos comandos mostrados no Código 3.9.

```
conda create -n <env_name> python=3.6 tensorflow=1.15
conda activate <env_name>
```

Código 3.9 – Criar ambiente conda

Para facilitar a utilização do tutorial desenvolvido neste trabalho, foi criado um repositório no *GitHub* contendo a biblioteca e todos os arquivos necessários para a execução. Dessa forma, após a criação e ativação do ambiente, deve-se clonar o repositório <https://github.com/lucassouza4/TCC_2.git>. Dessa forma, após a criação e ativação do ambiente, deve-se clonar o repositório utilizando no terminal o comando exemplificado em 3.10. A disponibilização dos arquivos no repositório é importante para garantir a replicabilidade do trabalho desenvolvido.

```
git clone https://github.com/lucassouza4/TCC_2.git
```

Código 3.10 – Clonar repositório do *GitHub*

Após a preparação, é necessário iniciar a instalação. Para evitar conflitos de dependência, recomenda-se instalar as versões específicas no código 3.11 para os pacotes *cmake*, *opencv-python* e *dm-tree*. Após a instalação desses pacotes, é necessário instalar as dependências mencionadas no arquivo *requirements.txt*. Por fim, deve-se instalar a biblioteca de aprendizado federado da IBM. Para isso, execute os seguintes comandos dentro da pasta clonada:

```
pip3 install cmake==3.25.0
pip3 install opencv-python==4.5.3.56
```

```
pip3 install dm-tree==0.1.8
pip3 --no-cache-dir install -r requirements.txt
pip3 install federated-learning-lib/federated_learning_lib-*-py3-none-
    any.whl
```

Código 3.11 – Instalação da biblioteca juntamente com suas dependências

Ao retornar ao terminal, após a instalação, caso esteja utilizando uma Máquina Virtual (VM), é necessário configurar o Jupyter Notebook de acordo com as diretrizes apresentadas no código 3.12, a fim de permitir o acesso através do navegador da máquina na qual o SSH está sendo executado. Além disso, é crucial expor o Jupyter Notebook, bem como abrir as portas 8888 e 5000 da VM, permitindo que outras máquinas possam se conectar (no caso deste tutorial, foi utilizada a plataforma *Oracle Coud*).

```
jupyter notebook --generate-config
nano ~/.jupyter/jupyter_notebook_config.py
c.NotebookApp.ip = '*'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 8888

sudo firewall-cmd --zone=public --permanent --add-port=8888/tcp
sudo firewall-cmd --zone=public --permanent --add-port=5000/tcp
sudo firewall-cmd --reload
```

Código 3.12 – exposição do Jupyter Notebook nas VMs

3.3.3 Criação do Notebook para o agregador e participantes

Para a configurar o agregador, é preciso definir um modelo. Foi fornecida uma estrutura de exemplo, como apresentado no Código 3.3. Para adaptá-la ao modelo que deseja utilizar, é necessário alterar os campos *num_classes*, *img_rows* e *img_cols*, as camadas do modelo e a biblioteca utilizada para realizar o treinamento. No exemplo apresentado no tutorial, a biblioteca *Keras* foi utilizada juntamente com a base de dados especificada na Seção 3.1.

Após a especificação do modelo a ser utilizado, é necessária a definição das configurações do agregador (Código 3.5). O agregador contém informações de conexão referentes à máquina de execução e seu arquivo de configuração contém informações sobre o algoritmo de fusão, hiperparâmetros do treinamento e protocolo. Para utilizar o notebook, é necessário alterar as seguintes informações para adaptação aos dados e algoritmos utilizados no treinamento:

- *Connection*: Esse campo contém informações como o endereço IP e a porta da máquina agregadora, além do tipo de conexão. No exemplo a seguir, o Flask foi utilizado como tipo de conexão.

- *Data*: Esse campo contém informações sobre os dados a serem trabalhados, como o caminho do arquivo *.npz* da base de dados e o manipulador específico para a leitura dessas informações.
- *Fusion*: Esse campo especifica qual dos métodos de fusão será utilizado.
- *hyperparams*: Existem dois tipos de hiperparâmetros: globais e locais. Os hiperparâmetros globais definem o tempo máximo de conexão, a quantidade de participantes do treinamento, o quórum mínimo de participantes para que um treinamento seja iniciado, a quantidade de épocas globais e a acurácia mínima para que o treinamento seja interrompido. Já os hiperparâmetros locais definem a quantidade de épocas locais e o valor do otimizador.

Diferentemente do treinamento realizado anteriormente utilizando o terminal da máquina, o treinamento no notebook requer a instância de um agregador, passando as configurações geradas anteriormente ao executar a célula 3.13. Em seguida, devem ser definidas as métricas para a coleta de informações 3.14. Após a conexão dos participantes ser estabelecida, o treinamento iniciará automaticamente, garantindo a precisão e a confiabilidade dos resultados obtidos.

```
from ibmfl.aggregator.aggregator import Aggregator
aggregator = Aggregator(config_dict=agg_config)

aggregator.start()
```

Código 3.13 – Instânciação do agregador

```
#1 Initialize the metrics collector variables

num_parties = agg_config['hyperparams']['global']['num_parties']
eval_party_accuracy = [[] for _ in range(num_parties)]
iterations = [[] for _ in range(num_parties)]

#2 Register handler for metrics collector

def get_metrics(metrics):
    keys = list(metrics['party'].keys())
    keys.sort()
    for i in range(len(keys)):
      eval_party_accuracy[i].append(metrics['party'][keys[i]]['acc'])
      iterations[i].append(metrics['fusion']['curr_round']*agg_config['
   hyperparams']['local']['training']['epochs'])

mh = aggregator.fusion.metrics_manager
mh.register(get_metrics)

#3 start the training

```

```
21 aggregator.start_training()
```

Código 3.14 – geração dos hiperparâmetros e início do treinamento

Ao término do treinamento, é imprescindível que o agregador seja desligado, a fim de encerrar todas as conexões de maneira adequada. Essa medida é fundamental para evitar problemas como vazamento de memória ou outros tipos de falhas.

```
1 aggregator.stop()
```

Código 3.15 – Interrupção do agregador

No que se refere aos participantes do treinamento, é necessário seguir o processo descrito na seção 3.2.1.3. Caso algum modelo já conhecido pela biblioteca seja utilizado, é possível utilizar o arquivo *generate_data.py*, localizado na pasta *examples*. Para executar esse arquivo, basta executar a célula 3.16. Como parâmetros para a geração dos dados, é necessário especificar o número de participantes, a partir do qual serão criados arquivos separados para cada parte, o nome da base de dados para que o código disponibilizado pela ferramenta possa identificar a base utilizada e, por fim, a quantidade de dados por arquivo.

```
1 %run examples/generate_data.py -n $num_parties -d $dataset -pp 200
```

Código 3.16 – geração dos dados

Embora a ferramenta em questão ofereça suporte a algumas bases de dados, para este estudo foram utilizados dados médicos, sendo necessário realizar o pré-processamento dos dados de forma individualizada. Considerando os testes que serão realizados, foram incluídos dois códigos no notebook do participante para facilitar a separação dos dados.

O primeiro código, presente no trecho 3.17, tem como objetivo dividir o arquivo de dados em partes iguais, de acordo com a quantidade de participantes envolvidos no treinamento federado. Já o segundo código, exemplificado em 3.18, realiza a separação dos dados de acordo com as classes presentes na base de dados. Com essa configuração, será possível avaliar os impactos que um possível viés dos dados pode trazer para o treinamento federado. É importante destacar que cada máquina terá acesso apenas aos dados de uma das classes pertencentes ao conjunto.

Dessa forma, a distribuição homogênea e não homogênea dos dados será avaliada para entender como ela pode afetar o desempenho do modelo de treinamento federado. A etapa de pré-processamento dos dados é crucial para garantir que os mesmos estejam adequados para a análise e que as conclusões tiradas sejam confiáveis. Por isso, é importante seguir rigorosamente os procedimentos de pré-processamento para garantir que os dados estejam bem estruturados e prontos para serem utilizados na análise.

```
1 import numpy as np
2
3 class DivisorArquivoNpz:
```

```
    def __init__(self, nome_arquivo, num_partes):
        self.nome_arquivo = nome_arquivo
        self.num_partes = num_partes

    def dividir(self):
        data = np.load(f'{self.nome_arquivo}.npz')

        partes = {}
        for chave, valor in data.items():
            partes[chave] = np.array_split(valor, self.num_partes)

        for i in range(self.num_partes):
            nome_arquivo = f'{self.nome_arquivo}_{i}.npz'
            np.savez(nome_arquivo, **{chave: valor[i] for chave, valor
   in partes.items()})
```

Código 3.17 – geração dos dados de forma igualitária

```
import numpy as np

data = np.load('./pneumoniamnist.npz')

x_train = data['train_images'] # array com os dados
y_train = data['train_labels'] # array com as etiquetas

x_test = data['test_images'] # array com os dados
y_test = data['test_labels'] # array com as etiquetas

x_val = data['val_images'] # array com os dados
y_val = data['val_labels'] # array com as etiquetas

classes_train = np.unique(y_train)
classes_test = np.unique(y_test)
classes_val = np.unique(y_val)

for classe in classes_train:

    idx_classe_train = np.where(y_train == classe)[0]
    dados_classe_train = x_train[idx_classe_train]
    etiquetas_classe_train = y_train[idx_classe_train]

    idx_classe_test = np.where(y_test == classe)[0]
    dados_classe_test = x_test[idx_classe_test]
    etiquetas_classe_test = y_test[idx_classe_test]

    idx_classe_val = np.where(y_val == classe)[0]
    dados_classe_val = x_val[idx_classe_val]
    etiquetas_classe_val = y_val[idx_classe_val]
```

```
31
32      np.savez(f'dados_{classe}.npz',
33           train_images=dados_classe_train, train_labels=
    etiquetas_classe_train,
34           test_images=dados_classe_test, test_labels=
    etiquetas_classe_test,
35           val_images=dados_classe_val, val_labels=etiquetas_classe_val)
```

Código 3.18 – geração dos dados por classe

Inicialmente, para trabalhar com dados que não são conhecidos pela biblioteca, foi necessário adicionar ao notebook do participante um manipulador para que a base de dados pudesse ser interpretada. Em seguida, é preciso executar a configuração dos participantes, conforme exemplificado nos códigos 3.7 e 3.8. No entanto, ao contrário do código apresentado anteriormente, no notebook o nome do arquivo npz, o nome do manipulador e o caminho devem ser definidos como variáveis que serão configuradas na primeira célula do notebook.

Para realizar o treinamento, foi adicionado o código 3.19, em que é criada uma instância da classe Party, passando como parâmetro as configurações de treinamento determinadas anteriormente. Em seguida, assim como no treinamento realizado através do terminal, é necessário registrar o participante junto ao agregador, especificando seu endereço IP externo para conexão.

```
1 from ibmfl.party.party import Party
2 import tensorflow as tf
3
4 party_config = get_party_config(party_id)
5 party = Party(config_dict=party_config)
6 party.start()
7 party.register_party()
8 party.proto_handler.is_private = False  ## allows sharing of metrics
    with aggregator
9 party.proto_handler.connection.source_info['ip'] = '3.145.98.236'
```

Código 3.19 – geração dos dados por classe

4 Experimentos e Resultados

Neste capítulo são apresentados, interpretados e analisados todos os resultados alcançados no trabalho.

4.1 Instalação

O aprendizado federado é uma área relativamente nova dentro da inteligência artificial comparado com outros ramos. Por isso, a quantidade de conteúdo disponível na internet ainda é muito limitado. Dificultando assim, não só a instalação mas também a utilização das diversas ferramentas existentes. Todavia, com a documentação disponibilizada pela própria IBM, fica fácil e simples de realizar a instalação da ferramenta. Como visto em seções anteriores, tanto no site da ferramenta quanto no GitHub, existem tutorias para auxiliar na implantação e alguns conhecimento básico acerca da ferramenta.

A diferença de implantação existente se encontra no fato de que, para implantação em serviços de *cloud*, mudou-se a configuração do IP local para o IP interno das respectiva máquinas nos Códigos 3.5, 3.7 e 3.8. Além dessas mudanças, para adicionar uma camada de segurança ao treinamento, utilizou-se o Código 3.2 para configurar o uso do *broker* disponibilizado pela própria IBM. Todavia, a utilização do mesmo é opcional, apenas a troca de IP é o suficiente para realizar o treinamento de forma remota.

4.2 Testes com a base de dados TissueMNIST

Em um primeiro momento, para testar a instalação da ferramenta, realizou-se alguns experimentos afim de mensurar a complexidade, viabilidade e eficiência de um treinamento federado de forma simples. Visto que, a aferição de forma mais apurada dos resultados do treinamento seria tratada futuramente. Os testes realizados de forma empírica tiveram como objetivo analisar como o número de épocas globais e locais podem afetar o desempenho do aprendizado federado e não em obter os melhores resultados. Tais experimentos tinham como objetivo gerar questionamentos para embasar estudos futuros.

No que diz respeito a configuração, foram realizados testes variando a quantidade de épocas globais e locais, conforme mostrado na Tabela 4.1. No treinamento federado, cada uma das duas máquinas utilizadas realizaram os testes sobre metade da base. Já no treinamento centralizado, seguiu-se o número total de épocas do experimento distribuído, porém com a base completa. Ou seja, se foram realizadas três épocas locais e cinco épocas globais tem-se no total quinze épocas. Dessa forma, o treinamento local foi executado com um total de 15 épocas.

Épocas globais	Épocas locais	Total
1	1	1
1	5	5
2	1	2
2	5	10
3	1	3
3	5	15
4	1	4
4	5	20
5	1	5
5	5	25

Tabela 4.1 – Sumário dos testes realizados sobre a base *TissueMNIST* com o *Federated Learning*.
Fonte: Elaborado pelo autor.

Baseado na Tabela 4.2 é visível a diferença no tempo de execução dos treinamentos. Uma vez que, no aprendizado federado, cada uma das máquinas trabalha com metade da base de dados e de forma paralela. Ou seja, a base de dados *TissueMNIST* possui pouco mais de 165 mil imagens que no treinamento local foram avaliadas de forma sequencial em um único treinamento. Enquanto que, no treinamento federado cada parte executou em paralelo o mesmo treinamento. Porém, com pouco mais de 80 mil imagens.

Épocas globais	Épocas locais	Épocas totais	*Federated Learning* (Segundos)	ML Tradicional (Segundos)
1	1	1	206	425
1	5	5	992	2338
2	1	2	400	926
2	5	10	1989	4545
3	1	3	397	1403
3	5	15	3073	6918
4	1	4	859	1876
4	5	20	4374	9246
5	1	5	975	2320
5	5	25	5259	11543

Tabela 4.2 – Tempo de execução do treinamento em segundos em relação ao número de épocas totais.
Fonte: Elaborado pelo autor.

O simples fato de dividir a base de dados, diminuindo a quantidade de imagem que cada parte deve processar, reduziu o tempo de execução de uma única época de 425 segundos para 206 segundos no pior caso. Como pode ser observado na primeira linha da Tabela 4.2. Contudo, como exemplificado na Tabela 4.3, apesar da melhora no tempo de execução, houve uma piora na acurácia quando executamos apenas uma época global. Tal resultado pode significar que, o treinamento federado pode se tornar vantajoso apenas quando executado com um número maior

de épocas globais. A explicação para tal hipótese pode ter ligação com a quantidade de imagens analisadas por instância, que dependendo da disposição dos dados, pode enviesar os resultados e pelo fato de não agregar os resultados obtidos é como se executássemos de forma isolada treinamentos com apenas metade dos dados conhecidos.

Épocas globais	Épocas locais	Épocas totais	*Federated Learning* (%)	ML Tradicional (%)
1	1	1	40,82	47,59
1	5	5	54,97	57,08
2	1	2	51,05	56,77
2	5	10	60,31	57,98
3	1	3	55,57	54,09
3	5	15	63,68	59,68
4	1	4	55,05	55,23
4	5	20	66,42	61,05
5	1	5	55,54	55,56
5	5	25	69,15	61,83

Tabela 4.3 – Acurácia do treinamento em relação ao número de épocas totais.
Fonte: Elaborado pelo autor.

Épocas globais	Épocas locais	Épocas totais	*Federated Learning*	ML Tradicional
1	1	1	1,6480	1,2792
1	5	5	1,2158	1,1817
2	1	2	1,3296	1,1833
2	5	10	1,0792	1,1385
3	1	3	1,1944	1,2455
3	5	15	0,9770	1,0964
4	1	4	1,2129	1,2222
4	5	20	0,9012	1,0615
5	1	5	1,1973	1,2054
5	5	25	0,9395	1,0429

Tabela 4.4 – Perda do treinamento em relação ao número de épocas totais.
Fonte: Elaborado pelo autor.

Após a criação dos notebooks, com o intuito de testá-los e avaliar novas configurações de treinamento, foram realizados novos experimentos. Esses experimentos tiveram como base a diversificação na disposição dos dados, um número maior de testes executados e a utilização em conjunto de quatro máquinas.

Inicialmente, os experimentos realizados não foram repetidos várias vezes para aferir uma média dos resultados. Por esse motivo, cada um dos experimentos a seguir foi executado dez vezes, a fim de validar os resultados obtidos. Além disso, o foco inicial voltou-se para a variação do número de épocas globais e locais. Dessa vez, variou-se também a disposição dos dados. Foram realizados testes dividindo de forma igualitária os dados, mas também foram realizados

testes criando um viés nos dados trabalhados ao separá-los por classe. Por exemplo, se tivéssemos duas máquinas e duas classes, cada máquina ficaria com todos os dados de uma determinada classe.

Dessa forma, será possível observar como o aprendizado federado se comporta ao trabalhar com dados mais próximos da realidade, onde temos hospitais infantis e hospitais que atendem principalmente adultos, e onde, no final, ambos farão parte do mesmo treinamento.

A Tabela 4.5 apresenta resultados de um experimento de treinamento com divisão homogênea dos dados. Pode-se observar que a configuração com 10 épocas totais e 3 participantes obteve a melhor acurácia, com um valor de 75,47% de acurácia. Além disso, essa configuração também teve a maior perda, indicando que o modelo está se ajustando melhor aos dados. A configuração com 10 épocas totais e 2 participantes também apresentou uma boa acurácia, com um valor de 73,68% de acurácia. No entanto, essa configuração apresentou uma perda um pouco menor do que a configuração com 3 participantes.

Épocas totais	Participantes	Perda	Acurácia (%)
1	2	3,8213	73,51
10	2	4,2614	73,68
1	3	3,3531	70,97
10	3	4,3576	75,47

Tabela 4.5 – Treinamento com divisão homogênea.
Fonte: Elaborado pelo autor.

A configuração com apenas 1 época total apresentou uma acurácia um pouco menor do que as outras configurações, indicando que o modelo precisa de mais épocas para se ajustar melhor aos dados. Além disso, essa configuração apresentou uma perda menor do que as outras configurações, o que sugere que o modelo ainda não está se ajustando bem aos dados.

Em resumo, a Tabela 4.6 indica que a configuração com 10 épocas totais e 3 participantes apresentou os melhores resultados em termos de acurácia e perda, o que pode indicar que o aumento do número de participantes acompanhado de um aumento do número de épocas pode trazer benefícios ao modelo.

A Tabela 4.6 apresenta os resultados de um experimento de treinamento com divisão heterogênea de dados. Ao analisar os dados apresentados na tabela, observa-se que o aumento do número de épocas totais resultou em uma diminuição da perda e um aumento na acurácia, o que é esperado em um processo de treinamento de modelos. Adicionalmente, a inclusão de mais participantes no processo de treinamento também levou a melhorias na acurácia, embora os efeitos sobre a perda tenham sido mais variados.

Épocas totais	participantes	Perda	Acurácia (%)
1	2	1.6861	89,13
10	2	1.6205	89,62
1	3	0.3005	95,39
10	3	0.4279	96,97

Tabela 4.6 – Treinamento com divisão heterogênea.
Fonte: Elaborado pelo autor.

No entanto, é importante destacar que outros fatores podem ter influenciado os resultados apresentados na Tabela 4.6, tais como a arquitetura do modelo, o conjunto de dados utilizado e o método de divisão dos mesmos. Dessa forma, é necessário ter cautela ao generalizar esses resultados para outras situações.

Em síntese, a Tabela 4.6 apresenta resultados de um experimento de treinamento com divisão heterogênea de dados. Entretanto, é preciso ter em mente que a melhoria observada pode ter sido influenciada pelas características específicas do conjunto de dados, que é binário, ou seja, possui apenas duas classes e de tamanho moderado. Ao dividir os dados por classe, o modelo pode ter se ajustado melhor aos dados.

4.3 *Federated Learning* em dispositivos móveis

Durante a fase de experimentação, foram realizados testes para verificar a viabilidade da utilização dos notebooks em dispositivos móveis. No entanto, nenhuma solução satisfatória foi encontrada tanto para dispositivos IOS quanto para dispositivos Android.

Durante a busca por soluções, foram pesquisados aplicativos nas respectivas lojas que pudessem simular ambientes Linux ou executar notebooks, porém, todos possuíam versões fixas de python e outras dependências, o que inviabilizou a instalação. Foi feita uma consulta sobre a possibilidade de utilização da ferramenta em dispositivos móveis no canal oficial do Slack da IBM, contudo, a resposta obtida foi que o suporte é limitado para dispositivos móveis. Dessa forma, a utilização dos tutoriais em dispositivos móveis foi postergada. No entanto, sugere-se que o suporte para dispositivos móveis seja uma área de estudo para trabalhos futuros.

5 Considerações Finais

O aprendizado federado é uma área relativamente nova dentro da inteligência artificial, comparada com outros ramos. Por isso, a quantidade de conteúdo disponível na internet ainda é limitada, dificultando não só a instalação, mas também a utilização das diversas ferramentas existentes. Todavia, com a documentação disponibilizada pela própria IBM, torna-se fácil e simples realizar a instalação da ferramenta. Como visto em seções anteriores, tanto no site da ferramenta quanto no GitHub, existem tutoriais para auxiliar na implantação e em alguns conhecimentos básicos acerca da ferramenta.

No estudo realizado sobre as ferramentas PySyft e IBM Federated Learning, foi possível concluir que, pela maturidade de ambas as ferramentas, a IBM Federated Learning se mostrou mais próxima do nosso objetivo de encontrar uma ferramenta simples e de fácil utilização por parte dos cientistas de dados. Isso se deve tanto à facilidade de implantação quanto à variedade de métodos de agregação e algoritmos suportados.

Apesar das facilidades oferecidas pela ferramenta em questão, sua utilização inicial é bastante manual. Isso se deve à necessidade de configuração da base de dados, codificação dos arquivos para interpretação e leitura de dados, bem como à execução dos comandos da ferramenta diretamente no terminal das máquinas. Com a finalidade de facilitar o processo de treinamento federado, este trabalho apresenta a criação de notebooks que reúnem todas as etapas necessárias em um único lugar. Isso inclui desde a divisão dos dados até a execução propriamente dita, de forma a permitir que modificações posteriores sejam realizadas de maneira simples e rápida.

Por fim, foram executados treinamentos com diversas configurações de épocas, a fim de medir de forma simples a eficiência do treinamento federado em relação à acurácia, tempo de execução e perda, em comparação com o treinamento local. Para testar novas hipóteses e configurações de treinamento, foram realizados novos experimentos com base na diversificação na disposição dos dados, um número maior de testes executados e a utilização em conjunto de máquinas reais e virtuais. Os experimentos foram executados dez vezes para validar os resultados obtidos. Além disso, no primeiro momento, o foco voltou-se para a variação do número de épocas globais e locais. Posteriormente, focou-se na disposição variada dos dados, sendo realizados testes dividindo de forma igualitária os dados, mas também criando um viés nos dados trabalhados ao separá-los por classe. Dessa forma, foi possível observar como o aprendizado federado se comporta ao trabalhar com dados mais próximos da realidade, onde temos hospitais infantis, hospitais que atendem mais adultos e que, no final, ambos farão parte de um mesmo treinamento.

Além disso, ao utilizar máquinas com configurações diversas, percebeu-se que, uma vez que a execução de novas épocas depende de todos os modelos globais, ter uma grande variação de configurações de hardware pode trazer problemas ao desempenho do treinamento, visto que

as máquinas mais lentas podem ser o gargalo da execução, desperdiçando assim os recursos alocados nas máquinas de melhor configuração.

Referências

BBC. *Entenda o escândalo de uso político de dados que derrubou valor do Facebook e o colocou na mira de autoridades*. 2018. Disponível em: <https://g1.globo.com/economia/tecnologia/noticia/entenda-o-escandalo-de-uso-politico-de-dados-que-derrubou-valor-do-facebook-e-o-colocou-na-mira-de-au ghtml>. Acesso em: 03 de Novembro 2022.

BUDRIONIS, A.; MIARA, M.; MIARA, P.; WILK, S.; BELLIKA, J. G. Benchmarking pysyft federated learning framework on mimic-iii dataset. *IEEE Access*, IEEE, v. 9, p. 116869–116878, 2021.

CARBONELL, J. G.; MICHALSKI, R. S.; MITCHELL, T. M. 1 - an overview of machine learning. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Ed.). *Machine Learning*. San Francisco (CA): Morgan Kaufmann, 1983. p. 3–23. ISBN 978-0-08-051054-5. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780080510545500054>.

Cem Dilmegani. *What is Federated Learning? Techniques, Use Cases & Benefits*. 2022. Disponível em: <https://research.aimultiple.com/federated-learning/>. Acesso em: 17 de outubro 2022.

GITHUB. 2022. Disponível em: <https://github.com/OpenMined/PySyft>. Acesso em: 10 de outubro 2022.

HALL, A. J.; JAY, M.; CEBERE, T.; CEBERE, B.; VEEN, K. L. van der; MURARU, G.; XU, T.; CASON, P.; ABRAMSON, W.; BENAISSA, A.; SHAH, C.; ABOUDIB, A.; RYFFEL, T.; PRAKASH, K.; TITCOMBE, T.; KHARE, V. K.; SHANG, M.; JUNIOR, I.; GUPTA, A.; PAUMIER, J.; KANG, N.; MANANNIKOV, V.; TRASK, A. Syft 0.5: A platform for universally deployable structured transparency. *CoRR*, abs/2104.12385, 2021. Disponível em: <https://arxiv.org/abs/2104.12385>.

IBM. *Frameworks, fusion methods, and Python versions*. 2022. Disponível em: <https://www.ibm.com/docs/en/cloud-paks/cp-data/4.5.x?topic=learning-frameworks-fusion-methods-python-versions>. Acesso em: 17 de outubro 2022.

IBM. *O que é um broker de serviços?* 2022. Disponível em: <https://www.ibm.com/docs/en/cloud-paks/cp-data/4.5.x?topic=models-federated-learning>. Acesso em: 11 de outubro 2022.

IBM. *Setup da ferramenta da IBM para aprendizado federado*. 2022. Disponível em: <https://github.com/IBM/federated-learning-lib/blob/main/setup.md>. Acesso em: 11 de outubro 2022.

Kim Martineau. *What is federated learning?* 2022. Disponível em: <https://research.ibm.com/blog/what-is-federated-learning>. Acesso em: 10 de outubro 2022.

LEE, G. H.; SHIN, S.-Y. Federated learning on clinical benchmark data: Performance assessment. *J Med Internet Res*, v. 22, n. 10, p. e20891, Oct 2020. ISSN 1438-8871. Disponível em: <http://www.jmir.org/2020/10/e20891/>.

LUDWIG, H.; BARACALDO, N.; THOMAS, G.; ZHOU, Y.; ANWAR, A.; RAJAMONI, S.; ONG, Y. J.; RADHAKRISHNAN, J.; VERMA, A.; SINN, M.; PURCELL, M.; RAWAT, A.; MINH, T. N.; HOLOHAN, N.; CHAKRABORTY, S.; WITHERSPOON, S.; STEUER, D.; WYNTER, L.; HASSAN, H.; LAGUNA, S.; YUROCHKIN, M.; AGARWAL, M.; CHUBA, E.; ABAY, A. IBM federated learning: an enterprise framework white paper V0.1. *CoRR*, abs/2007.10987, 2020. Disponível em: <https://arxiv.org/abs/2007.10987>.

Nasron Cheong. *Identifying the right use cases for federated learning and analytics*. 2022. Disponível em: <https://www.integrate.ai/blog/identifying-the-right-use-cases-for-federated-learning-pftl>. Acesso em: 17 de outubro 2022.

PYSYFT. 2021. Disponível em: <https://docs.openmined.org/pysyft/packages/syft/examples/duet>. Acesso em: 10 de outubro 2022.

RedHat. *O que é um broker de serviços?* 2022. Disponível em: <https://www.redhat.com/pt-br/topics/cloud-native-apps/service-brokers>. Acesso em: 11 de outubro 2022.

RISTANI, E.; SOLERA, F.; ZOU, R.; CUCCHIARA, R.; TOMASI, C. Performance measures and a data set for multi-target, multi-camera tracking. In: HUA, G.; JÉGOU, H. (Ed.). *Computer Vision – ECCV 2016 Workshops*. Cham: Springer International Publishing, 2016. p. 17–35. ISBN 978-3-319-48881-3.

SUN, T.; LI, D.; WANG, B. Decentralized federated averaging. *CoRR*, abs/2104.11375, 2021. Disponível em: <https://arxiv.org/abs/2104.11375>.

Théo Ryffel. *Part 11 - Secure Deep Learning Classification*. 2022. Disponível em: <https://notebook.community/OpenMined/PySyft/examples/tutorials/Part%2011%20-%20Secure%20Deep%20Learning%20Classification>. Acesso em: 17 de outubro 2022.

WEI, K.; LI, J.; DING, M.; MA, C.; YANG, H. H.; FAROKHI, F.; JIN, S.; QUEK, T. Q. S.; POOR, H. V. Federated learning with differential privacy: Algorithms and performance analysis. *CoRR*, abs/1911.00222, 2019. Disponível em: <http://arxiv.org/abs/1911.00222>.

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.

Apêndices

APÊNDICE A – Notebook de instalação

Este apêndice contém o notebook de instalação utilizado neste trabalho. O notebook foi criado para facilitar a instalação de todas as bibliotecas e pacotes necessários para a execução dos notebooks de análise de dados.

O Notebook de Instalação pode ser acessado em: <https://github.com/lucassouza4/TCC_2/blob/main/instalacao.ipynb>

APÊNDICE B – Notebook do agregador

Este apêndice contém o notebook Aggregator utilizado neste trabalho. O notebook foi criado para orquestrar todo o treinamento federado.

O Notebook Aggregator pode ser acessado em: <https://github.com/lucassouza4/TCC_2/blob/main/Aggregator.ipynb>

APÊNDICE C – Notebook do participante

Este apêndice contém o notebook Participant utilizado neste trabalho. O notebook foi criado para realização dos experimento e visualização dos resultados.

O Notebook Participant pode ser acessado em: <https://github.com/lucassouza4/TCC_2/blob/main/participant.ipynb>