`Flower-PROV`: Captura Distribuída de Dados de Proveniência em Experimentos de Aprendizado Federado*

Camila Lopes[1], Alan L. Nunes[1], Cristina Boeres[1], Lúcia M. A. Drummond[1], Daniel de Oliveira[1]

[1]Instituto de Computação - Universidade Federal Fluminense
Av. Gal. Milton Tavares de Souza, s/nº, Boa Viagem – Niterói/RJ – Brasil

{camila_ol, alan_lira}@id.uff.br {boeres, lucia, danielcmo}@ic.uff.br

Abstract. *Federated Learning (FL) is a distributed technique that allows multiple users to train models collaboratively without accessing private or sensitive data. The training of a model requires multiple iterations, and the duration depends on the several configurations set, e.g., hyperparameters. Analyzing hyperparameters during the FL workflow allows for a better understanding of the model and opens up opportunities for improvement. This paper introduces an FL framework named* `Flower-PROV`*, which aims to capture provenance data during training to track configurations and metrics, enabling the analysis of hyperparameters in real time. The provenance database follows the W3C PROV recommendation.*

Resumo. *O Aprendizado Federado (AF) é uma técnica descentralizada que possibilita que vários usuários treinem modelos de Aprendizado de Máquina de forma colaborativa, sem precisar acessar dados privados ou sensíveis. O treinamento de um modelo pode exigir várias iterações, e a duração de cada iteração depende diretamente das configurações definidas, como os valores dos hiperparâmetros. Analisar os hiperparâmetros durante o treinamento permite uma melhor compreensão do modelo treinado e abre oportunidades para melhorias. Este artigo apresenta o arcabouço* `Flower-PROV` *que tem como objetivo capturar dados de proveniência durante o treinamento para rastrear configurações e métricas de avaliação, possibilitando a análise dos hiperparâmetros em tempo real. O banco de dados de proveniência segue a recomendação W3C PROV, facilitando a comparação, explicação e reprodução desses experimentos.*

1. Introdução

Na última década, o campo de Aprendizado de Máquina tem recebido considerável atenção da comunidade acadêmica e de empresas. Esse interesse foi impulsionado principalmente pela disponibilidade abundante de dados para treinamento e pela disseminação do Aprendizado Profundo (*Deep Learning*) [Goodfellow et al. 2016], uma técnica fundamental para o reconhecimento de padrões em larga escala. No entanto, apesar dos avanços nessas técnicas, os *workflows* de treinamento de modelos exigem que todos os dados estejam acessíveis, o que pode se tornar problemático devido a preocupações com a *privacidade*. Muitos conjuntos de dados contêm informações sensíveis que, se expostas, podem representar riscos

*Video da demonstração da ferramenta disponível em `https://youtu.be/-dV2QT7wfAc`. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Os autores gostariam ainda de agradecer ao CNPq (*grant* 311898/2021-1) e FAPERJ (*grant* E-26/202.806/2019) pelo apoio financeiro.

significativos. Embora existam técnicas de anonimização de dados, como a Privacidade Diferencial [Dwork 2006], nem sempre é possível obter modelos eficientes.

Com o intuito de preencher essa lacuna entre o treinamento de modelos e a preocupação com a privacidade, surgiu o conceito de Aprendizado Federado (AF) [McMahan et al. 2017]. O AF é uma abordagem distribuída que possibilita a colaboração de diversos clientes (*i.e.*, *workers*) no treinamento de modelos, ao mesmo tempo em que mantém seus dados de treinamento privados, sem a necessidade de expor informações sensíveis. O modelo final é obtido por meio de iterações (*i.e.*, *rounds*), em que cada cliente treina seu modelo localmente e envia uma versão atualizada para o servidor, que agrega os modelos treinados por cada cliente e devolve os parâmetros atualizados ao fim da iteração.

Apesar da evolução do AF nos últimos anos [Li et al. 2020], ainda se fazem necessários mecanismos que permitam a análise e o monitoramento do treinamento de um modelo. Cada iteração do *workflow* usando AF pode durar de minutos a várias horas, já que o tempo de treinamento e a qualidade do modelo treinado em cada iteração estão diretamente associados ao conjunto de dados de entrada e à escolha de valores de hiperparâmetros. Uma escolha ruim pode levar a um desperdício de tempo e recursos, uma vez que o usuário só é capaz de avaliar o resultado no fim do treinamento. Uma opção interessante é analisar os valores das métricas de avaliação (*e.g.*, acurácia) após cada iteração e, eventualmente, tomar ações como interromper o treinamento e reiniciá-lo com novos valores de hiperparâmetros.

Os dados de proveniência [Freire et al. 2008] se apresentam como uma escolha natural para registrar caminho de derivação dos dados ao longo do *workflow* de treinamento de um modelo, fornecendo uma visão global dos dados com suas dependências. Por meio dos dados de proveniência, o usuário é capaz de consultar e analisar como os dados são produzidos em cada iteração, consultando valores de hiperparâmetros, métricas de avaliação, *etc*. Embora o uso de proveniência para compreensão de treinamento de modelos não seja algo novo [Chapman et al. 2022, Pina et al. 2023], ainda existem poucas propostas no contexto do AF [Peregrina et al. 2022]. Entretanto, as soluções propostas ou não capturam o histórico de derivação completo (focando apenas em atribuição de responsabilidades , ou seja, nos agentes que executam as atividades do *workflow* e não nas atividades em si), ou possuem representações proprietárias para os dados que não são aderentes a padrões de representação de proveniência como o W3C PROV [Groth and Moreau 2013].

Neste artigo de demonstração, apresentamos uma extensão do arcabouço *Flower* [Beutel et al. 2020], denominada `Flower-PROV`, que tem como objetivo facilitar a gerência e consulta de dados de proveniência em experimentos de AF. O arcabouço foi desenvolvido com foco em aplicações *Cross-Silo* (onde o número de clientes é reduzido). O `Flower-PROV` permite a captura automática e distribuída de dados de proveniência retrospectiva (*r-prov*) e prospectiva (*p-prov*), que incluem informações como valores de hiperparâmetros, acurácia, *etc*. Para avaliação, realizamos experimentos de treinamento de um modelo de classificação com base no conjunto de dados CIFAR-10 [Krizhevsky 2009]. A base de proveniência segue o padrão W3C PROV e permite que os usuários realizem consultas em tempo de execução, o que facilita a interpretação e análise dos resultados obtidos. O restante deste artigo está organizado da seguinte forma: na Seção 2, apresentamos a arquitetura do `Flower-PROV`; na Seção 3, descrevemos a demonstração realizada para ilustrar a utilização e benefícios do `Flower-PROV`; e, por fim, na Seção 4, apresentamos as conclusões e perspectivas futuras deste trabalho.

2. O Arcabouço `Flower-PROV`

Conforme mencionado anteriormente, o `Flower-PROV` estende o arcabouço `Flower` [Beutel et al. 2020] adicionando capacidade analítica do treinamento por meio da captura automática e distribuída de dados de proveniência, onde um servidor é responsável pela agregação do modelo e cada cliente treina modelos locais. A Figura 1 apresenta a arquitetura do `Flower-PROV`, onde os retângulos em cinza claro representam as extensões propostas ao arcabouço original.

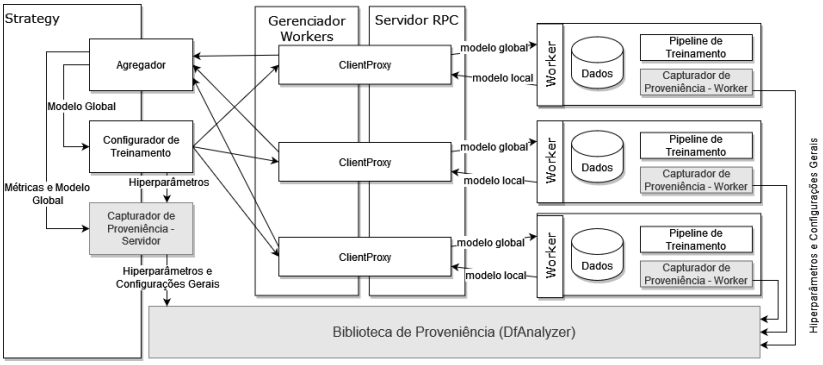

Figura 1. Arquitetura do `Flower-PROV`

A arquitetura do `Flower-PROV` é composta por cinco componentes principais: (i) *Strategy*, (ii) Gerenciador de Clientes, (iii) Servidor RPC, (iv) Clientes e (v) Biblioteca de Proveniência. O *Gerenciador de Clientes* é responsável por amostrar os clientes que tem como objetivo treinar os modelos locais. Em coordenação com o *Servidor RPC*, ele gerencia vários objetos *ClientProxy*, que são objetos associados a um *Cliente* conectado ao *Servidor*. Um *ClientProxy* é responsável por enviar e receber mensagens para cada um dos clientes, que realizam o treinamento do modelo local. Toda a orquestração do *workflow* de AF é realizada pelo componente *Strategy*, que prepara a próxima iteração de AF no servidor e envia as configurações para os clientes envolvidos no processo de treinamento. Após cada cliente gerar um novo modelo local, o *Strategy* recebe as atualizações, *e.g.*, pesos de uma rede neural, e agrega os resultados no componente *Agregador*. Após a agregação, o *Strategy* avalia o modelo agregado no componente *Configurador de Treinamento*, de acordo com as métricas de avaliação escolhidas (*e.g.*, acurácia). Todos os passos mencionados até o momento já faziam parte do arcabouço `Flower` original.

Em cada uma das etapas mencionadas anteriormente, os dados de proveniência são capturados por componentes específicos. O componente responsável por receber mensagens contendo os dados de proveniência do servidor e dos clientes e estruturá-los em uma forma que possa ser consultada é a *Biblioteca de Proveniência*. Na versão atual do `Flower-PROV`, a *Biblioteca de Proveniência* escolhida é uma extensão da biblioteca de proveniência DfAnalyzer [Silva et al. 2020], que fornece métodos genéricos para capturar e consultar dados de proveniência. Vale ressaltar que a DfAnalyzer foi estendida para ser acoplada ao `Flower-PROV`, pois não permitia a representação de ciclos, algo inerente ao processo de AF. O `Flower-PROV` envia os dados de proveniência capturados para a DfAnalyzer de maneira assíncrona, sem interferir no desempenho do experimento de AF.

No lado do servidor, o componente *Capturador de Proveniência - Servidor* captura metadados relacionados às etapas de agregação e avaliação, como as métricas de avaliação

e qual o modelo enviado aos clientes. As atividades e metadados do lado do cliente são coletados usando o componente *Capturador de Proveniência - Cliente*. É importante ressaltar que toda a captura nos clientes é realizada de forma distribuída. Todos os dados são armazenados pela *Biblioteca de Proveniência* em um banco de dados de proveniência compatível com o padrão W3C PROV e estão disponíveis em tempo de execução, *i.e.*, assim que são capturados, e podem ser consultados, o que permite utilizar esses dados para monitoramento. Na versão atual, o banco de dados de proveniência se encontra armazenado no SGBD colunar MonetDB[1]. O `Flower-PROV` tem código aberto e está disponível em `https://github.com/UFFeScience/Flower-PROV`.

3. Demonstração

A demonstração do `Flower-PROV` tem como estudo de caso a arquitetura de rede neural MobileNetV2 [Sandler et al. 2018], uma rede neural convolucional (CNN) de visão computacional. O conjunto de dados utilizado foi o CIFAR-10 [Krizhevsky 2009], um conjunto de dados multi-classe balanceado composto por imagens coloridas de 32×32 *pixels*, divididas em 10 classes, cada uma contendo 6.000 imagens. No total, foram utilizadas 50.000 imagens ($\approx$ 83,33%) para treinamento e 10.000 ($\approx$ 16,66%) para teste. Para simular o cenário que requer privacidade de dados no `Flower-PROV`, cada cliente recebeu uma partição balanceada e distinta do conjunto de dados original do CIFAR-10 gerada com a ferramenta *dataset-splitter*[2], desenvolvida por um dos autores. Além disso, a demonstração usa uma estratégia de agregação de pesos síncrona, com envio dos modelos locais ao servidor no final de cada iteração, o que se traduz em um overhead intrínseco de comunicação.

A demonstração se inicia com a apresentação dos *scripts* de execução do servidor e dos clientes. Neles, são definidos os pontos de captura de dados de proveniência onde são invocados os componentes *Capturadores de Proveniência*. Deve ser definido que tipos de dados são capturados (*i.e.*, proveniência prospectiva ou *p-prov*). Um exemplo é apresentado na Figura 2 onde o *template* dos dados que serão capturados na etapa de configuração do servidor são definidos. Podemos perceber que os parâmetros de interesse (*e.g.*, *server_id*, *num_rounds* e *enable_ssl*) são definidos e seus valores serão capturados durante o treinamento do modelo com o `Flower-PROV`. O usuário pode adicionar outros parâmetros conforme a necessidade de análise. Na demonstração planejada, consideramos como base um cenário com um servidor e três clientes.

```
tf1 = Transformation("ServerConfig")
    tf1_input = Set(
        "iServerConfig",
        SetType.INPUT,
        [
            Attribute("server_id", AttributeType.NUMERIC),
            Attribute("address", AttributeType.TEXT),
            Attribute("max_message_length_in_bytes", AttributeType.TEXT),
            Attribute("num_rounds", AttributeType.NUMERIC),
            Attribute("round_timeout_in_seconds", AttributeType.NUMERIC),
            Attribute("accept_rounds_with_failures", AttributeType.TEXT),
            Attribute("enable_ssl", AttributeType.TEXT),
            Attribute("enable_dynamic_adjustment", AttributeType.TEXT),
            Attribute("server_aggregation_strategy", AttributeType.TEXT),
            Attribute("fraction_fit", AttributeType.NUMERIC),
            Attribute("fraction_evaluate", AttributeType.NUMERIC),
            Attribute("min_fit_clients", AttributeType.NUMERIC),
            Attribute("min_evaluate_clients", AttributeType.NUMERIC),
            Attribute("min_available_clients", AttributeType.NUMERIC),
        ],
    )

```

Figura 2. Definição de Proveniência Prospectiva (*p-prov*) do `Flower-PROV`

Na demonstração, o treinamento da MobileNetV2 é realizado variando os valores de dois hiperparâmetros: (i) *batch size* $bs = \{32, 64, 128, 256, 512\}$ e número de épocas

[1] `https://www.monetdb.org/`

[2] dataset-splitter - `https://github.com/alan-lira/dataset-splitter`

(*epochs*) do cliente $e = \{1, 2, 3, 4, 5\}$. Todos os outros hiperparâmetros foram mantidos em seus valores padrão (*e.g.*, 100 iterações). Durante o treinamento da CNN, os dados de proveniência são coletados e armazenados para consulta. O usuário pode então escolher entre (i) visualizar o grafo de proveniência (Figura 3(a)), onde cada transformação executada durante o treinamento (*e.g.*, agregação dos modelos locais, treinamento do modelo local, *etc*) e valores de hiperparâmetros podem ser visualizados, e (ii) submeter consultas a base de proveniência que possam ser usadas como insumo para elaboração de gráficos de acompanhamento, como por exemplo a evolução da acurácia do modelo ao longo das iterações. A consulta em SQL apresentada na Figura 4 retorna os valores de acurácia e perda por iteração do treinamento e da validação. O valor da acurácia por iteração é usado então para criar o gráfico apresentado na Figura 3(b). A partir dele, o usuário é capaz de observar possíveis discrepâncias na acurácia, o que pode indicar necessidade de ajustar outros hiperparâmetros no próximo treinamento, como o número de épocas.

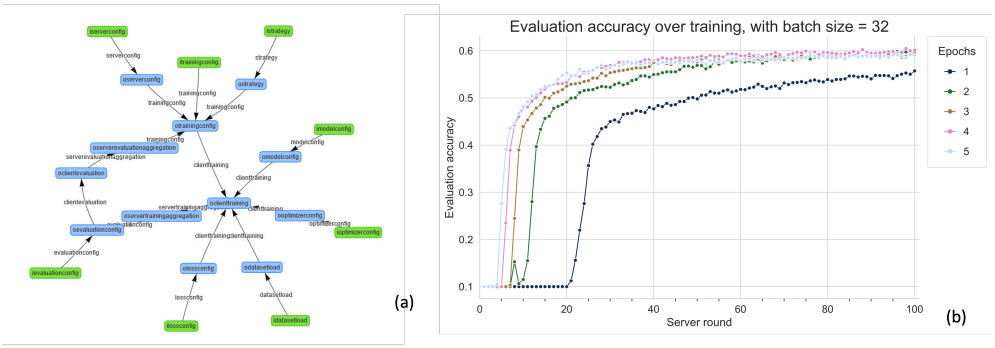

Figura 3. (a) Grafo que representa as transformações de dados realizadas durante o treinamento (b) Gráfico que apresenta a evolução da Acurácia por *round*

```
SELECT st.server_round, st.accuracy, st.loss,
se.accuracy, se.loss, se.evaluation_time,
FROM oservertrainingaggregation as st
JOIN oserverevaluationaggregation as se
ON st.server_round = se.server_round;
```

Figura 4. Consulta submetida à base de proveniência para extração de métricas

4. Conclusão

O treinamento de modelos de Aprendizado de Máquina geralmente requer que todos os dados de treinamento estejam prontamente disponíveis. No entanto, essa disponibilidade irrestrita de dados para análise e descoberta de padrões pode apresentar riscos significativos de violação de privacidade, especialmente quando dados sensíveis estão envolvidos. A anonimização de dados é uma abordagem comumente aplicada para reduzir esse risco, mas não é uma tarefa simples e pode não gerar modelos eficientes. Para mitigar a chance de vazamento de dados sensíveis, o conceito de Aprendizado Federado (AF) foi proposto.

O AF permite que os usuários treinem seus modelos localmente, e posteriormente, um servidor agrega esses modelos locais em um modelo global. No entanto, monitorar e compreender o *workflow* de treinamento em um experimento de AF não é uma tarefa trivial. Este artigo de demonstração apresenta o arcabouço `Flower-PROV`, que tem como objetivo auxiliar a análise de configurações e adaptações de hiperparâmetros no treinamento de experimentos de AF por meio da captura distribuída de dados de proveniência.

O `Flower-PROV` permite a captura desses dados em cada cliente envolvido no treinamento, possibilitando consultas em tempo de execução e armazenamento em um banco de dados que segue o padrão W3C PROV. Além disso, ao adotar o PROV, o `Flower-PROV` é capaz de exportar os dados de proveniência em um formato que permite a comparação com resultados obtidos em outros arcabouços de AF que também seguem o padrão W3C PROV.

Referências

Beutel, D. J. et al. (2020). Flower: A Friendly Federated Learning Research Framework. *arXiv preprint arXiv:2007.14390*.

Chapman, A., Lauro, L., Missier, P., and Torlone, R. (2022). DPDS: assisting data science with data provenance. *Proc. VLDB Endow.*, 15(12):3614–3617.

Dwork, C. (2006). Differential privacy. *33rd ICALP 2006, Proceedings, Part II*, volume 4052, pages 1–12. Springer.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering*, 10(3):11–21.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Groth, P. and Moreau, L. (2013). W3C PROV - An Overview of the PROV Family of Documents. Available at `https://www.w3.org/TR/prov-overview/`.

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. (2020). Federated Optimization in Heterogeneous Networks. *Proceedings of Machine Learning and Systems (MLSys)*. mlsys.org.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. *Proc. of 20th AISTATS*, pages 1273–1282.

Peregrina, J. A., Ortiz, G., and Zirpins, C. (2022). Towards a Metadata Management System for Provenance, Reproducibility and Accountability in Federated Machine Learning. *Advances in Service-Oriented and Cloud Computing*, pages 5–18. Springer.

Pina, D., Chapman, A., Oliveira, D., and Mattoso, M. (2023). Deep learning provenance data integration: a practical approach. *IPAW*, pages 1542–1550. ACM.

Sandler, M., A. Howard, M. Z., Zhmoginov, A., and Chen, L. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

Silva, V., Campos, V., Guedes, T., Camata, J., de Oliveira, D., Coutinho, A. L., Valduriez, P., and Mattoso, M. (2020). Dfanalyzer: Runtime dataflow analysis tool for computational science and engineering applications. *SoftwareX*, 12:100592.