Building and Evaluating Federated Models for Edge Computing

Yasaman Amannejad
Department of Mathematics and Computing
Mount Royal University, Calgary, Canada
Email: yamannejad@mtroyal.ca

***Abstract*—Today's state-of-the-art machine learning (ML) techniques, such as deep learning (DL) networks are typically trained using cloud platforms, leveraging elastic scalability of the cloud. For such processing, data from various sources need to be transferred to a cloud server. While this works well for some application domains, it is not suitable for all applications due to concerns about latency, connectivity, and privacy. For example, sharing life logging photos and videos from cellphones and wearable devices can cause privacy concerns for users, and transferring the unstructured data can burden the communication network. With the increase of such applications, federated learning (FL) is proposed as a distributed ML solution for learning on edge devices, such as cellphones and wearable devices. In FL, clients collaboratively train a model on their device without sharing their data. Each client trains a local model with their data and shares the model parameters with a FL server to aggregate and build a global model. Shifting from traditional ML techniques to federated solutions requires comparing these two approaches. Moreover, users need to study the performance of FL models to decide if federation is feasible for their learning task. In this paper, we propose an automated solution to compare centrally trained DL models with federated solutions. The tool allows users to easily analyze the accuracy of federated models for their learning task and study the effect of the federated parameters. We show the features of our tool building central and federated DL models from an input model structure for recognizing images in the MNIST benchmark dataset.**

***Index terms*— Distributed Machine Learning, Federated Learning, Edge Computing.**

I. Introduction

Learning from distributed data over edge devices has gained increasing interest in recent years. This increase in popularity is because of the following reasons. *First*, there is a need for processing a massive amount of data that is continuously being generated through devices such as cellphones, wearable devices, and autonomous vehicles. Transferring all data to a remote cloud server is not always feasible due to limiting factors such as the network bandwidth of these devices and the long propagation delays that can incur unacceptable latency. *Second*, the private nature of the data, e.g., life-logging videos and recorded phone calls, causes privacy concerns for sharing the data with cloud servers. *Finally*, transferring such massive data to a cloud server for processing can burden the backbone networks especially for applications with unstructured data such as images and videos. These factors along with the improvements in the storage and computation capacity of edge devices are shifting the data processing and model training of ML applications from cloud servers to edge devices.

978-3-903176-31-7

Federated learning (FL) is introduced recently as a decentralized ML approach suitable for edge computing [1]. In FL, data remains private on devices. Clients work cooperatively to train a complex ML model without sharing their data. Consider a large number of client devices, such as cellphones, each with personal collections of photos. If all the distributed data on these devices are accessible in a central place, one can obtain a high-performance ML model that is trained on an extremely large dataset. However, it is not desirable for clients to share their data due to privacy concerns. Each device trains a local model using its own private data and shares the model parameters, e.g., model weights, with an FL server that aggregates them into a global model. This process continues iteratively in multiple rounds until a desirable accuracy is achieved or a training budget is hit.

FL has recently gained popularity and is shown to be effective in different application areas [2–7]. To facilitate building and evaluating FL applications, open-source algorithms and protocols [8–10], simulators and libraries [11–14], and benchmark datasets [13] are proposed by others. However, they do not provide an automated solution for evaluating federated models. Practitioners and researchers can benefit from an automated solution that allows them to study the effect of the federated parameters and compare federated and central models. Such a solution can help users to decide if federation is an appropriate approach for their learning task.

To address this need, we propose AutoEdgeML, an automated tool for building federated ML solutions. Specifically, users can input their model structure and the tool provides them with templates for training the model in federated and central settings. The tool also allows users to study the effect of federated parameters on the model accuracy and fine-tune the values of the parameters. When a federated dataset is not available, AutoEdgeML allows users to define their data partitioning strategies to create distributed datasets for clients from an existing central benchmark dataset. To show how AutoEdgeML works, we build a federated DL model for an image recognition task on MNIST benchmark [15] and compare its accuracy with the same model when trained centrally. The results show that the federated model built using the tool is able to recognize images with high accuracy comparable with the central solution, and the tool provides insights on the selection of federated parameters.

II. Federated Learning

Federated learning (FL) is an ML setting where many clients (e.g. mobile devices) collaboratively train a model under the

coordination of a central server while keeping the training data decentralized. There are two main entities in FL process: the data owners, i.e., $client$, and the global model owner, i.e., $server$. Let $Clients = \{client_1, client_2, ..., client_n\}$ denote the set of n clients, each of which has a dataset D_i, $i \in n$, stored on their personal devices, e.g., cellphones. In classical approaches, all clients send their data to a server and the server trains a conventional model, $model$, using all data $D = \cup_{i=1}^{n} D_i$. However, as described earlier, this is not possible for all applications due to privacy concerns, and also the network bandwidth limitations of the edge computing devices. In FL, clients do not share their private data with the remote server, instead, they build a local model, $model_i$, using their own data, D_i, and share the parameters of their trained model with the FL server. The server collects all local model parameters and aggregates them to build a global model, $model_{federated}$. This global model is sent back to clients for further enhancements. This process continues for r rounds.

A typical training process is shown in Fig. 1 and includes the following four steps:

1) **Model broadcast**: The server defines the structure of the model to be trained by all clients and decides on the hyper-parameters of the local and global models, e.g., the optimizer or the learning rate of a neural network. Next, the server broadcasts the model and the hyper-parameters to clients. In the first round, the broadcasted model is not trained. In the following rounds, the server broadcasts the model aggregated from the local training of clients.
2) **Local model training**: Clients participating in the training process download the broadcasted model and train the model based on their local data. The objective of each client during its local training is to find optimal model parameters that minimize the model loss based on the client's local data.
3) **Model parameter updates**: When each client, $client_i$, finishes its training, they share their trained model, $model_i$, with the FL server.
4) **Global model aggregation**: When the participating clients share their models with the server, the server aggregates them into a global model, $model_{global}$. The server's objective is to minimize a global loss function.

Training an FL model is an iterative process and happens in multiple rounds r. In each round of the training process, a subset of all clients, $C \subset Clients$ send their local updates to the server. The number of training rounds and participating clients can affect the accuracy of the global model. In an ideal case, the accuracy of the model trained in this distributed and iterative format should be close to the accuracy of the model trained in conventional format over all available data in a central place. However, the FL model may result in lower accuracy as a trade-off for gaining higher data privacy. Therefore, it is important to evaluated federated models and compare their accuracy with central solutions. We propose an automated solution for evaluating the models trained in central and federated settings and studying the effect of federated parameters on the performance of FL models.

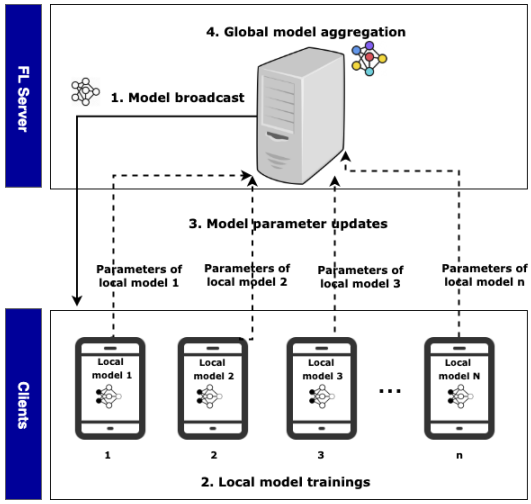

Fig. 1. The main steps in FL training process

The distribution of data among clients is an important factor that can affect the performance of federated models. When a conventional benchmark dataset is available, data needs to be distributed among clients before training an FL model. In general, there are two main forms of data distribution: IID[1], and non-IID data. IID data means that each batch of data used for a client's local model update is statistically identical to a uniformly drawn sample with replacement from the entire training dataset, i.e., the union of all local datasets at the clients. In practice, clients collect their training data independently which results in having training datasets with different sizes and distributions from other clients. IID assumption does not always hold in federated settings. In this work, we support both IID and non-IID dataset creations. This allows users to study the performance of federated models under various data distribution scenarios.

III. Related work

FL framework is proposed by McMahan et al. [1] as a distributed ML solution with a focus on keeping data private. The growing demand for FL technology has resulted in a number of open-source algorithms and protocols [8–10], simulator and libraries [11, 12, 14], and benchmark datasets [13].

NVIDIA Clara [8], PaddleFL [9] and FATE [10] are open-source frameworks that support FL. NVIDIA Clara is a healthcare application framework for imaging that makes it easy for developers to build, manage, and deploy intelligent medical imaging workflows and instruments. This platform supports training collaborative FL models for medical use cases. PaddleFL allows users to deploy an FL system in distributed clusters. FATE [10] is an open-source project initiated by Webank's AI Department to provide a secure computing

[1]Independent and Identically Distributed

framework to support the federated AI ecosystem. It implements multiple secure computation protocols in compliance with data protection regulations.

To help users build their own federated models, programming libraries [12, 14], simulation tools [11, 13, 14], and benchmark datasets are provided [13]. PySyft [12] is a Python library for secure, private DL. PySyft decouples private data from model training using FL, differential privacy, and multi-party computation within PyTorch. Mugunthan et al. [11] has proposed a simulator that supports simulating the accuracy of FL models and the convergence time of the model. Leaf [13] provides multiple datasets, as well as simulation and evaluation capabilities. TensorFlow Federated [14] provides a programming framework as well as simulation capabilities that target research use cases. Our tool is built on top of the TensorFlow Federated library and provides an automated solution for building and evaluating FL models.

IV. Features of AutoEdgeML

AutoEdgeML allows users to create FL models from an input model structure and study the effect of the federated parameters on the performance of FL models. Users can compare the performance of their FL model with a model trained in conventional central fashion. For this purpose, AutoEdgeML supports data partitioning strategies, model creation, and model evaluation. The tool is built on top of well-known ML libraries such as Keras and TensorFlow, and supports models created by these libraries. Users can input their model structures and the tool will create the template of the FL process to train the model and observe the effect of the federated parameters. The tool generates federated models based on TensorFlow Federated library. In cases where a federated dataset is not available, the tool supports converting central datasets into IID and non-IID datasets to train and test federated models. The code used in this work is publicly available [16].

A. Data Partitioning

As described in Section II, each client in federated models owns their own data and the distribution of the data may vary among clients. The distribution of the data can affect the training time and the accuracy of the trained models. Therefore, when comparing a conventional solution with a federated solution, users need to consider this in their comparison. AutoEdgeML allows users to generate IID and non-IID datasets for federated clients from their central datasets. For generating non-IID data, AutoEdgeML allows users to define their custom data partitioning strategy. A custom strategy may assign a subset of target labels or feature values to each client.

B. Building Model

AutoEdgeML can create the template for training an input model in federated and conventional settings. The input models can be in one of the two accepted model formats: HDF5 and SavedModel. HDF5 is a grid format for storing multi-dimensional arrays of numbers and is suitable for storing model weights and structure. SavedModel format is another way to serialize models and is the default format in TensorFlow 2.x.

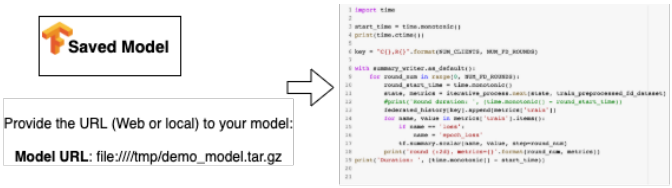

Fig. 2. Building model using AutoEdgeML

C. Model Evaluation

When the data and the model are converted into federated format, the next step for users is to evaluate the accuracy of the federated model under various parameters and compare it with the accuracy of the model trained centrally. For a fair comparison between central and federated models, the tool uses the same hyper-parameters for both models. The FL model has additional parameters that can affect the model accuracy. The tool allows users to study the performance of the model with different number of clients and training rounds. Tensorboard is integrated into the tool to allow users to view the results of their study. Moreover, users can benefit from the custom graphs provided for comparing the effect of the FL parameters. Sample graphs are shown in Fig. 3

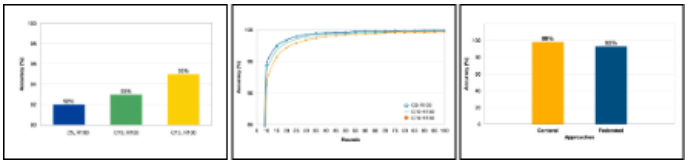

Fig. 3. Generated graphs for model evaluation

V. Experiment Setup and Results

To show how AutoEdgeML works, we use the tool to build a federated model based on a DL model structure for an image recognition task. The architecture of the network is shown in Fig. 4. Using AutoEdgeML we generate an FL structure based on a central deep network with convolution, pooling, and dense layers. To make a fair comparison between the federated and central models, we use the same hyper-parameters, e.g., batch size and the number of epoch for both models. We use MNIST dataset [17] which is a writer-annotated handwritten digit classification dataset collected from many users. In total, MNIST dataset contains $70,000$ grayscale images of size 28×28 for 10 hand-written digits.

We use AutoEdgeML on the Google Colab platform to build a federated model based on an input model structure and study the accuracy of the federated model in comparison with the central model. We first compare the two models when 5 clients participate in the training of the federated model for 100 rounds. Next, we show the performance of the federated model with various number of clients and rounds.

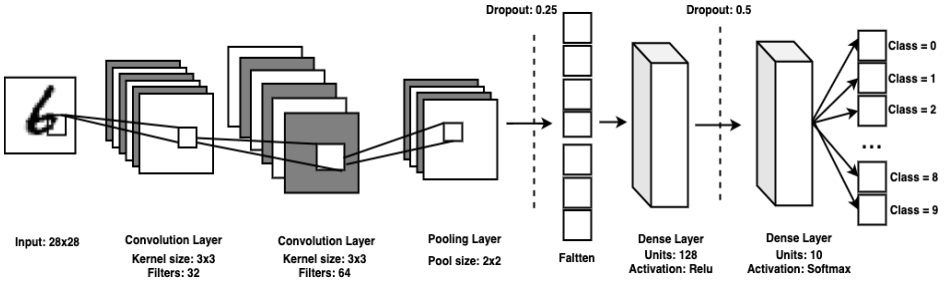

Fig. 4. The architecture of the model used in the study

A. Comparing Federated and Central Models

The FL model generated and trained by AutoEdgeML has achieved 92% accuracy when predicting for an unseen test dataset. Fig. 5 compares the accuracy of the central and federated models when trained with the same training data and the same hyper-parameters. As it can be seen in the figure, the federated model achieves lower accuracy compared to the central model. This lower accuracy is due to the fact that in a federated setting the data is distributed over the client devices and we do not have access to all data to train the global model. Instead, the global model is built by aggregating local models of the 5 clients who participated in the model training. The accuracy of the FL model can be improved by tuning the number of participating clients or the number of training rounds. Next, we study the effect of these two parameters.

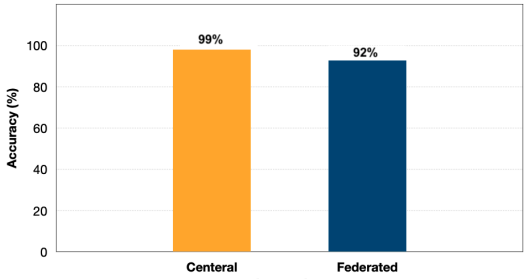

Fig. 5. Test accuracy of central and federated models

B. Effect of the Federated Parameters

Using AutoEdgeML, we evaluate the model accuracy with varying the number of rounds between 1 to 100 and the number of clients from 5 to 15 in steps of 5. Fig. 6 shows the accuracy of the federated models with different number of clients and training rounds. As it can be seen in Fig. 6, in all experiments the model converges and achieves high accuracy. The model trained with less number of clients quickly gets training accuracy close to 99%. This is while it takes more number of rounds for the model with 15 clients to achieve the same accuracy during the training time.

AutoEdgeML also reports the test accuracy of the models. The test accuracy shows the ability of the model to generalize and recognize unseen data records. This metric is often more important than the training accuracy. As it can be seen in

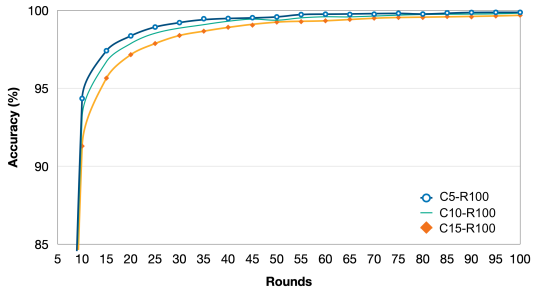

Fig. 6. Training accuracy in rounds with different number of clients

Fig. 7, the model trained with more number of clients, i.e. 15, achieves better accuracy on the test data. By comparing Fig. 6 and 7 we can see that incorporating more clients in the training process while may slow down the training process, it can result in a better model. A more comprehensive analysis on the effect of client selection is provided in our recent study [18].

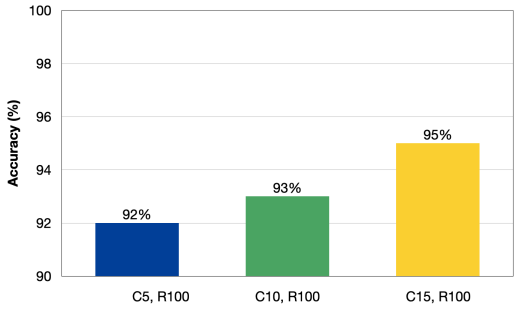

Fig. 7. Test accuracy with different number of clients

VI. Conclusion and Future Work

To help users and researchers study the performance of federated models for learning tasks with distributed datasets, this paper proposes an automated solution for building and evaluating FL models. AutoEdgeML is a tool that is designed for this purpose and this paper reports the progress on the early phases of the tool. We show the main features of the tool using one use case trained on MNIST benchmark. In the future, we plan to extend the features of the tool and also to evaluate the tool with more use cases.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

[2] E. Bakopoulou, B. Tillman, and A. Markopoulou, "A federated learning approach for mobile packet classification," *arXiv preprint arXiv:1907.13113*, 2019.

[3] Y. Liu, J. James, J. Kang, D. Niyato, and S. Zhang, "Privacy-preserving traffic flow prediction: A federated learning approach," *IEEE Internet of Things Journal*, 2020.

[4] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Federated learning for ultra-reliable low-latency v2v communications," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, IEEE, 2018.

[5] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, M. D. Mueck, and S. Srikanteswara, "Energy demand prediction with federated learning for electric vehicle networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[6] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot," in *IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 756–767, IEEE, 2019.

[7] A. Abeshu and N. Chilamkurti, "Deep learning: the frontier for distributed attack detection in fog-to-things computing," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 169–175, 2018.

[8] T. C. T. F. Authors, "Nvidia clara." https://developer.nvidia.com/clara, 2019. Accessed: 2020-07-20.

[9] T. P. Authors, "Paddlefl." https://github.com/PaddlePaddle/PaddleFL, 2019. Accessed: 2020-07-20.

[10] T. F. Authors, "Federated ai technology enabler." https://www.fedai.org/, 2019. Accessed: 2020-07-20.

[11] V. Mugunthan, A. Peraire-Bueno, and L. Kagal, "Privacyfl: A simulator for privacy-preserving and secure federated learning," *arXiv preprint arXiv:2002.08423*, 2020.

[12] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[13] S. Caldas, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[14] "Tensorflow federated learning." https://github.com/tensorflow/federated. Accessed: 2020-06-30.

[15] Y. LeCun, C. Cortes, and C. J.C. Burges, "Mnist dataset." http://yann.lecun.com/exdb/mnist/. Accessed: 2020-07-20.

[16] Y. Amannejad, "Federated learning tools." https://github.com/Yasaman-A/federated-learning-tools, 2020. Accessed: 2020-10-15.

[17] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," 2010.

[18] S. Lameh, W. Noble, Y. Amannejad, and A. Afshar, "Analysis of federated learning as a distributed solution for learning on edge devices," in *The International Conference on Intelligent Data Science Technologies and Applications (IDSTA2020)*.