
ON-DEVICE FEDERATED LEARNING WITH FLOWER

Akhil Mathur^{1,2} Daniel J. Beutel^{1,3} Pedro Porto Buarque de Gusmão⁴ Javier Fernandez-Marques⁴
Taner Topal^{1,3} Xinchi Qiu¹ Titouan Parcollet⁵ Yan Gao¹ Nicholas D. Lane¹

ABSTRACT

Federated Learning (FL) allows edge devices to collaboratively learn a shared prediction model while keeping their training data on the device, thereby decoupling the ability to do machine learning from the need to store data in the cloud. Despite the algorithmic advancements in FL, the support for on-device training of FL algorithms on edge devices remains poor. In this paper, we present an exploration of on-device FL on various smartphones and embedded devices using the Flower framework. We also evaluate the system costs of on-device FL and discuss how this quantification could be used to design more efficient FL algorithms.

1 INTRODUCTION

We have seen remarkable progress in enabling the execution of deep learning models on mobile and embedded devices to infer user contexts and behaviors (Warden & Situnayake, 2019; Fromm et al., 2018; Chowdhery et al., 2019; Malekzadeh et al., 2019; Lee et al., 2019). This has been powered by the increasing computational abilities of edge devices as well as novel software optimizations to enable cloud-scale models to run on resource-constrained devices. However, when it comes to the training of these edge-focused models, a working assumption has been that the models will be trained centrally in the cloud, using training data aggregated from several users.

Federated Learning (FL) (McMahan et al., 2017) aims to enable distributed edge devices (or users) to collaboratively *train* a shared prediction model while keeping their personal data private. At a high level, this is achieved by repeating three basic steps: i) local parameters update to a shared prediction model on each edge device, ii) sending the local parameter updates to a central server for aggregation, and iii) receiving the aggregated model back for the next round of local updates.

A major bottleneck to FL research is the paucity of frameworks that support federated training of workloads on mobile and embedded devices. While several frameworks including Tensorflow Federated (Google, 2020; Abadi et al., 2016) and LEAF (Caldas et al., 2018) enable simulation of FL clients, they cannot be used to understand the training dynamics and compute the system costs of FL on edge devices.

¹University of Cambridge, UK ²Nokia Bell Labs, UK ³Adap, Germany ⁴University of Oxford ⁵Avignon Université, France. Correspondence to: Akhil Mathur <akhilmathurs@gmail.com>.

Edge devices exhibit significant heterogeneity in their software stack, compute capabilities, and network bandwidth. All these system-related factors, in combination with the choice of the FL clients and parameter aggregation algorithms, can impact the accuracy and training time of models trained in a federated setting.

In this paper, we present our exploration of on-device training of FL workloads on Android smartphones and Nvidia Jetson series embedded devices, using the Flower framework (Beutel et al., 2020). Flower offers a stable implementation of the core components of an FL system, and provides higher-level abstractions to enable researchers to experiment and implement new ideas on top of a reliable stack. We first demonstrate how we use the language-, platform- and ML framework-agnostic capabilities of Flower to support on-device training of FL workloads on edge devices with heterogeneous hardware and software stacks. We then deploy these FL clients on various embedded devices as well as on Android smartphones hosted in the Amazon AWS Device Farm (<https://aws.amazon.com/device-farm/>). Finally, we present an evaluation to compute various system-related metrics of FL and highlight how this quantification could lead to the design of more efficient FL algorithms.

2 RELATED WORK

McMahan et al. (2017) introduced the basic federated averaging (FedAvg) algorithm and evaluated it in terms of communication efficiency. The optimization of distributed training with and without federated concepts has been covered from many angles (Jia et al., 2018; Chahal et al., 2018). Bonawitz et al. (2019) detail the design of a large-scale Google-internal FL system. TFF (Google, 2020), PySyft (Ryffel et al., 2018), LEAF (Caldas et al., 2018), FedML (He et al., 2020) are other open-source frameworks that support research and experimentation of FL workloads.

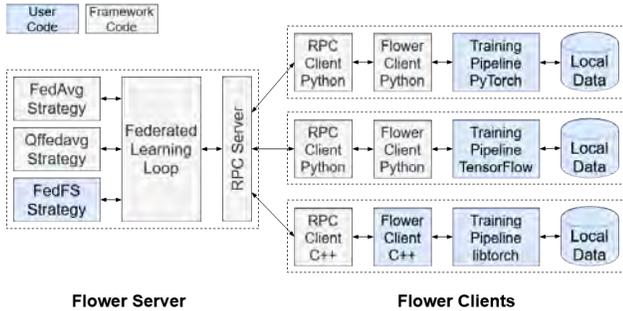


Figure 1. Flower framework architecture.

Given their relative hardware limitations, most of the works involving machine learning on mobile devices were originally aimed at adapting existing models to specific latency and storage constraints. For this purpose, optimized versions of TensorFlow and PyTorch were developed (David et al., 2020; Paszke et al., 2019) and can now be considered mainstream. Federated Learning, on the other hand, shifts the training burden from server to the client, which in turn creates the need for developing the adequate supporting back-ends (e.g. back-propagation) for low-power hardware as well. Recent works towards this goal include using low-precision training (Sun et al., 2020), controlled updates of biases to reduce memory (Cai et al., 2020), and early-exit models that provide a trade-off between accuracy and compute (Leontiadis et al., 2021). In time, as hardware capabilities in edge devices improve, we expect to see these advances being deployed by mainstream frameworks.

3 PRIMER ON FLOWER

Flower is a novel client-agnostic federated learning framework. One of the underlying design goals of Flower is to enable integrating with an inherently heterogeneous and ever-evolving edge device landscape. There are multiple dimensions of on-device heterogeneity, amongst them are operating systems, machine learning frameworks, programming languages, connectivity, and hardware accelerators.

The Flower core framework, shown in Figure 1, implements the infrastructure to run these heterogeneous workloads at scale. On the server side, there are three major components involved: the FL loop, the RPC server, and a (user customizable) *Strategy*. Strategy here refers to the federated averaging algorithms (e.g., FedAvg) used for aggregating the model parameters across clients. Clients connect to the RPC server which is responsible for monitoring these connections and for sending and receiving *Flower Protocol* messages. The FL loop is at the heart of the FL process: it orchestrates the learning process and ensures that progress is made. It does not, however, make decisions about *how* to proceed, those decisions are delegated to the currently configured *Strategy* implementation.

A distinctive property of this architecture is a server which

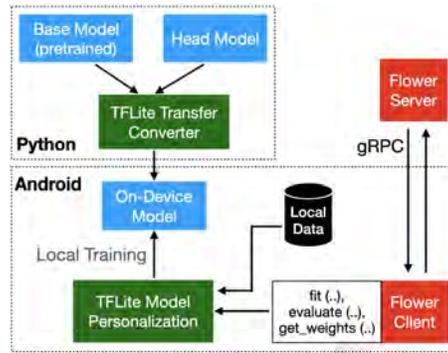


Figure 2. Flower Android client architecture.

is unaware of the nature of connected clients. This allows to train models across heterogeneous client platforms and implementations, including workloads comprised of different client-side ML frameworks. Furthermore, on-device training and evaluation can be implemented in different programming languages, a property especially important for research on mobile and embedded platforms. These platforms often do not support Python, but rely on specific languages (Java on Android, Swift on iOS) for idiomatic development, or native C/C++ for some embedded devices. Flower achieves a fully language-agnostic interface by offering protocol-level integration. The *Flower Protocol* defines core server-side messages such as *fit* and *evaluate*, which include the (serialized) global model parameters and expect return messages from the client that return either updated model parameters (or gradients) or evaluation results. Each message contains additional user-customizable metadata that allows the server to control on-device hyper-parameters, for example, the number of on-device training epochs. Due to space constraints, we refer the reader to (Beutel et al., 2020) and <https://flower.dev/> for more details on Flower.

4 FLOWER CLIENTS AND ON-DEVICE TRAINING

In this section, we describe two instances of on-device federated learning with Flower. First, we present how Flower clients can be developed in Java and deployed on Android phones in the AWS Device Farm for federated model training. Next, we discuss the implementation of Flower clients in Python and their deployment on heterogeneous embedded devices such as Nvidia Jetson series and Raspberry Pi.

4.1 Java Flower Clients for Android Smartphones

By design, Flower is language-agnostic and can work with any ML framework on the FL client, which maximizes its ability to federate existing training pipelines. However, it also means Flower inherits the limitations of these frameworks that currently offer limited support for on-device training on Android devices.

In the absence of a full-fledged model training library for

Android, we employ the TensorFlow Lite (TFLite) Model Personalization support on Android to perform on-device federated learning. More specifically, as shown in Figure 2, we first obtain a pre-trained *Base Model* (e.g., MobileNetV2 without its top layers). The parameters of this model are frozen (i.e., not updated during training), and it is only used as a feature extractor. Next, we define a *Head Model* which corresponds to the task-specific classifier that we want to train using federated learning. The input to the *Head Model* are the features extracted from the *Base Model*, and its weights are randomly initialized. Finally, we use the TFLITE TRANSFER CONVERTOR to port the Base and Head models to TFLite and package them inside an Android application for on-device training.

Inside the Android application, FLOWER CLIENT is a class which coordinates with the TFLite Model Personalization libraries and implements the three core methods required for federated training with Flower, namely (i) `get_weights()`, which gets the current weights of the *Head Model* to support server-side aggregation requests, (ii) `fit()`, which updates the parameters of the Head Model through local training and (iii) `evaluate()`, which computes test loss on the local dataset and communicates it to the server.

The FLOWER CLIENT spawns a background thread and sets up bi-directional streaming RPC with the Flower server using the STREAMOBSERVER class. Upon receiving messages from the Flower server, the background thread calls the appropriate TFLite methods to update the parameters of the *Head Model* using the local training data (e.g., by optimizing a cross-entropy loss for classification tasks). The updated parameters are sent to the Flower Server where they are aggregated across all clients, before being sent back to the clients for the next round of training.

4.2 Python Flower Clients on Embedded Devices

Contrary to Android, on-device training on Python-enabled embedded devices can be performed using platform-specific ML frameworks (e.g., TensorFlow, PyTorch). However, one of the open challenges to deploy on-device training workloads arises due to the heterogeneity of computing resources present in such devices. For example, NVIDIA Jetson devices provide GPU acceleration, while Raspberry Pi is limited to CPU-only workloads. Implementing different versions of an FL client for each target platform can quickly become unpractical in real-world scenarios.

Flower overcomes this challenge by providing a platform-agnostic way of writing FL clients. Figure 3 illustrates the design of Flower Clients for a NVIDIA Jetson TX2 and a Raspberry Pi device. Despite the differences in these two embedded platforms, we can use the same code to develop Flower Clients, and run them inside a platform-specific Docker image, enabling the client code to access platform-

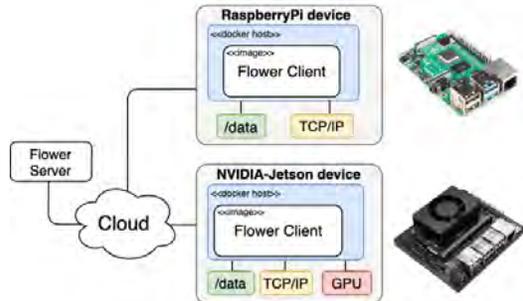


Figure 3. Flower clients can easily be deployed to heterogeneous devices by leveraging existing container-based frameworks (e.g. Docker) that interface with the host’s hardware.

Table 1. Android phones used from the AWS Device Farm

Device Name	Type	OS Version
Google Pixel 4	Phone	10
Google Pixel 3	Phone	10
Google Pixel 2	Phone	9
Samsung Galaxy Tab S6	Tablet	9
Samsung Galaxy Tab S4	Tablet	8.1.0

specific resources (e.g. GPU) when available.

More importantly, this platform-agnostic capability of Flower can be combined with its language-agnostic capabilities (i.e., support for Python, Java, C++ clients) to enable on-device training of federated workloads in highly-heterogeneous client setups, e.g., when the FL clients include Android phones, Android watches, Raspberry Pis, ARM micro-controllers, Nvidia Jetson devices etc.

5 EVALUATION

Our evaluation focuses on quantifying the system costs associated with running FL on various edge devices. In doing so, we also explore how such quantification could help FL developers to design novel algorithms that effectively trade-off between system costs and FL accuracy.

Datasets. Two datasets are used in our evaluation, namely *CIFAR-10* and *Office-31* (Office31, 2020), both of which are examples of object recognition datasets.

Deployment Setup. We run the Flower Server configured with the FedAvg strategy and host it on a cloud virtual machine. Two sets of edge devices are used in the evaluation: Android smartphones and Nvidia Jetson TX2 accelerators. To scale our experiments to a reasonably large number of Android clients with different OS versions, we deploy Flower Clients on the Amazon AWS Device Farm, which enables testing applications on real Android devices accessed through AWS. Table 1 list the smartphones from AWS Device Farm used in our evaluation. Nvidia Jetson TX2 devices support full-fledged PyTorch – this means we could successfully port existing PyTorch training pipelines to implement FL clients on them.

Table 2. Flower supports implementation of FL clients on any device that has on-device training support. Here we show various FL experiments on Android and Nvidia Jetson devices.

Local Epochs (E)	Accuracy	Convergence Time (mins)	Energy Consumed (kJ)
1	0.48	17.63	10.21
5	0.64	36.83	50.54
10	0.67	80.32	100.95

(a) Performance metrics with Nvidia Jetson TX2 as we vary the number of local epochs. Number of clients C is set to 10 and the model is trained for 40 rounds.

Number of Clients (C)	Accuracy	Convergence Time (mins)	Energy Consumed (kJ)
4	0.84	30.7	10.4
7	0.85	31.3	19.72
10	0.87	31.8	28.0

(b) Performance metrics with Android clients as we vary the number of clients. Local epochs E is fixed to 5 in this experiment and the model is trained for 20 rounds.

System Costs of FL. In Table 2, we present various performance metrics obtained on Nvidia TX2 and Android devices. First, we train a ResNet-18 model on the CIFAR-10 dataset on 10 Nvidia TX2 clients. In Table 2a, we vary the number of local training epochs (E) performed on each client in a round of FL. Our results show that choosing a higher E results in better FL accuracy, however it also comes at the expense of significant increase in total training time and overall energy consumption across the clients. While the accuracy metrics in Table 2a could have been obtained in a simulated setup, quantifying the training time and energy costs on real clients would not have been possible without a real on-device deployment. As reducing the energy and carbon footprint of training ML models is a major challenge for the community, Flower can assist researchers in choosing an optimal value of E to obtain the best trade-off between accuracy and energy consumption.

Next, we train a 2-layer DNN classifier (*Head Model*) on top of a pre-trained MobileNetV2 *Base Model* on Android clients for the Office-31 dataset. In Table 2b, we vary the number of Android clients (C) participating in FL, while keeping the local training epochs (E) on each client fixed to 5. We observe that by increasing the number of clients, we can train a more accurate object recognition model. Intuitively, as more clients participate in the training, the model gets exposed to more diverse training examples, thereby increasing its generalizability to unseen test samples. However, this accuracy gain comes at the expense of high energy consumption – the more clients we use, the higher the total energy consumption of FL. Again, based on this analysis obtained using Flower, researchers can choose an appropriate number of clients to find a balance between accuracy and energy consumption.

Table 3. Effect of computational heterogeneity on FL training times. Using Flower, we can compute a hardware-specific cutoff τ (in minutes) for each processor, and find a balance between FL accuracy and training time. $\tau = 0$ indicates no cutoff time.

	GPU ($\tau = 0$)	CPU ($\tau = 0$)	CPU ($\tau = 2.23$)	CPU ($\tau = 1.99$)
Accuracy	0.67	0.67	0.66	0.63
Training time (mins)	80.32	102 (1.27x)	89.15 (1.11x)	80.34 (1.0x)

Computational Heterogeneity across Clients. FL clients could have vastly different computational capabilities. While newer smartphones are now equipped with GPUs, other devices may have a much less powerful processor. How does this computational heterogeneity impact FL?

For this experiment, we use Nvidia Jetson TX2 as the client device, which has one Pascal GPU and six CPU cores. We repeat the experiment shown in Table 2a, but instead of using the embedded GPU for training, we train the ResNet-18 model on a CPU. In Table 3, we show that CPU training with local epochs $E = 10$ would take $1.27\times$ more time to obtain the same accuracy as the GPU training. This implies that even a single client device with low compute resources (e.g., a CPU) can become a bottleneck and significantly increase the FL training time.

Once we obtain this quantification of computational heterogeneity, we can design better federated optimization algorithms. As an example, we implement a modified version of FedAvg where each client device is assigned a cutoff time (τ) after which it must send its model parameters to the server, irrespective of whether it has finished its local epochs or not. This strategy has parallels with the FedProx algorithm (Li et al., 2018) which also accepts partial results from clients. However, the key advantage of using Flower is that we can compute and assign a processor-specific cutoff time for each client. For example, on average it takes 1.99 minutes to complete an FL round on the TX2 GPU. If we set the same time as a cutoff for CPU clients ($\tau = 1.99$ mins) as shown in Table 3, we obtain the same convergence time as GPU, at the expense of 3% accuracy drop. With $\tau = 2.23$, a better balance between accuracy and training time could be obtained on a CPU.

6 CONCLUSION

We presented our exploration of federated training of models on mobile and embedded devices by leveraging the language-, framework- and platform-agnostic capabilities of the Flower framework. We also presented early results quantifying the system costs of FL on the edge and its implications for the design of more efficient FL algorithms. Although our work is early stage, we hope it will trigger discussions at the workshop related to on-device training for FL and beyond.

REFERENCES

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.
- Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Parcollet, T., and Lane, N. D. Flower: A friendly federated learning research framework, 2020.
- Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C. M., Konečný, J., Mazzocchi, S., McMahan, B., Overveldt, T. V., Petrou, D., Ramage, D., and Roselander, J. Towards federated learning at scale: System design. 2019.
- Cai, H., Gan, C., Zhu, L., and Han, S. Tinytl: Reduce memory, not parameters for efficient on-device learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 11285–11297. Curran Associates, Inc., 2020.
- Caldas, S., Wu, P., Li, T., Konečný, J., McMahan, H. B., Smith, V., and Talwalkar, A. LEAF: A benchmark for federated settings. *CoRR*, abs/1812.01097, 2018. URL <http://arxiv.org/abs/1812.01097>.
- Chahal, K. S., Grover, M. S., and Dey, K. A hitchhiker’s guide on distributed training of deep neural networks. *CoRR*, abs/1810.11787, 2018. URL <http://arxiv.org/abs/1810.11787>.
- Chowdhery, A., Warden, P., Shlens, J., Howard, A., and Rhodes, R. Visual wake words dataset. *CoRR*, abs/1906.05721, 2019. URL <http://arxiv.org/abs/1906.05721>.
- David, R., Duke, J., Jain, A., Reddi, V. J., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., et al. Tensorflow lite micro: Embedded machine learning on tinymml systems. *arXiv preprint arXiv:2010.08678*, 2020.
- Fromm, J., Patel, S., and Philipose, M. Heterogeneous bitwidth binarization in convolutional neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18*, pp. 4010–4019, Red Hook, NY, USA, 2018. Curran Associates Inc.
- Google. Tensorflow federated: Machine learning on decentralized data. <https://www.tensorflow.org/federated>, 2020. accessed 25-Mar-20.
- He, C., Li, S., So, J., Zhang, M., Wang, H., Wang, X., Vepakomma, P., Singh, A., Qiu, H., Shen, L., Zhao, P., Kang, Y., Liu, Y., Raskar, R., Yang, Q., Annavaram, M., and Avestimehr, S. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.
- Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., Chen, T., Hu, G., Shi, S., and Chu, X. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *CoRR*, abs/1807.11205, 2018. URL <http://arxiv.org/abs/1807.11205>.
- Lee, T., Lin, Z., Pushp, S., Li, C., Liu, Y., Lee, Y., Xu, F., Xu, C., Zhang, L., and Song, J. Occlumency: Privacy-preserving remote deep-learning inference using sgx. In *The 25th Annual International Conference on Mobile Computing and Networking, MobiCom ’19*, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361699. doi: 10.1145/3300061.3345447. URL <https://doi.org/10.1145/3300061.3345447>.
- Leontiadis, I., Laskaridis, S., Venieris, S. I., and Lane, N. D. It’s always personal: Using early exits for efficient on-device cnn personalisation. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications, HotMobile ’21*, pp. 15–21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383233. doi: 10.1145/3446382.3448359. URL <https://doi.org/10.1145/3446382.3448359>.
- Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.
- Malekzadeh, M., Athanasakis, D., Haddadi, H., and Livshits, B. Privacy-preserving bandits, 2019.
- McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In Singh, A. and Zhu, X. J. (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR, 2017. URL <http://proceedings.mlr.press/v54/mcmahan17a.html>.
- Office31. Office 31 dataset. <https://people.eecs.berkeley.edu/~jhoffman/domainadapt/>, 2020. accessed 10-Oct-20.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8026–8037. Curran Associates, Inc., 2019.
- Ryffel, T., Trask, A., Dahl, M., Wagner, B., Mancuso, J., Rueckert, D., and Passerat-Palmbach, J. A generic framework for privacy preserving deep learning. *CoRR*, abs/1811.04017, 2018. URL <http://arxiv.org/abs/1811.04017>.
- Sun, X., Wang, N., Chen, C.-Y., Ni, J., Agrawal, A., Cui, X., Venkataramani, S., El Maghraoui, K., Srinivasan, V. V., and Gopalakrishnan, K. Ultra-low precision 4-bit training of deep neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1796–1807. Curran Associates, Inc., 2020.
- Warden, P. and Situnayake, D. *Tinymml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. ” O’Reilly Media, Inc.”, 2019.