

FLOWER: A FRIENDLY FEDERATED LEARNING FRAMEWORK

Daniel J. Beutel^{1,2} Taner Topal^{1,2} Akhil Mathur³ Xinchu Qiu¹ Titouan Parcollet⁴
 Pedro Porto Buarque de Gusmão¹ Nicholas D. Lane¹

ABSTRACT

Federated Learning (FL) has emerged as a promising technique for edge devices to collaboratively learn a shared prediction model while keeping training data on device, thereby decoupling the ability to do machine learning from the need to store data in the cloud. However, FL is difficult to implement and deploy in practice considering the heterogeneity in common edge device settings, e.g., different frameworks, languages, and hardware accelerators. On the systems side, progress has been two-fold: closed large-scale industrial systems running FL on diverse device fleets and open source research frameworks primarily used for single-machine simulation.

In this paper, we present *Flower*, a novel FL framework which unifies both perspectives. Flower is open source, supports heterogeneous environments including mobile and edge devices, and scales to a large number of distributed clients. Abstractions provided by Flower allow engineers to port existing workloads with little overhead, regardless of ML framework used, while also enabling researchers flexibility to experiment with novel approaches to advance the state-of-the-art. We describe the design goals and architecture of Flower and use it to evaluate the impacts of scale and heterogeneity on common FL methods in experiments with up to 1000 clients.

1 INTRODUCTION

There has been tremendous progress in enabling the execution of deep learning models on mobile and embedded devices to infer user contexts and behaviors (Fromm et al., 2018; Chowdhery et al., 2019; Malekzadeh et al., 2019; Lee et al., 2019; Yao et al., 2019; LiKamWa et al., 2016; Georgiev et al., 2017). This has been powered by the increasing computational abilities of mobile devices as well as novel algorithms which apply software optimizations to enable pre-trained cloud-scale models to run on resource-constrained devices. However, when it comes to the training of these mobile-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) is an emerging area of research in the machine learning community which 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,

¹Department of Computer Science and Technology, University of Cambridge, UK ²Adap, Hamburg, Hamburg, Germany ³UCLIC, University College London, UK ⁴Laboratoire Informatique d’Avignon, Avignon Université, France. Correspondence to: Daniel J. Beutel <daniel@adap.com>.

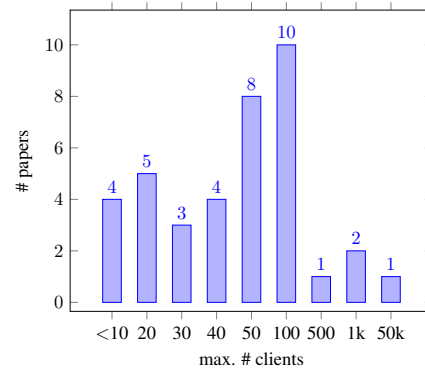


Figure 1. Survey of the number of FL clients used in FL research papers in the last two years. A vast majority of papers only use up to 100 clients. Appendix A.4 gives details of the papers considered.

and iii) receiving the aggregated model back for the next round of local updates.

From a systems perspective, a major bottleneck to FL research is the paucity of frameworks that support scalable execution of FL methods on mobile and edge devices. While few frameworks including Tensorflow Federated (Google, 2020; Abadi et al., 2016a) (TFF) and LEAF (Caldas et al., 2018a) enable experimentation on FL algorithms, they do not provide support for running FL on edge devices. System-related factors such as heterogeneity in the software stack, compute capabilities, and network bandwidth, affect model synchronization and local training. In combination with the choice of the client selection and parameter aggregation al-

gorithms, they can impact the accuracy and training time of models trained in a federated setting. The systems’ complexity of FL and the lack of scalable open-source frameworks can lead to a disparity between FL research and production. While closed production-grade systems report client numbers in the thousands or even millions (Hard et al., 2019), few research papers use populations of more than 100 clients, as can be seen in Table 1. Even those papers which have use than 100 clients rely on client simulations (e.g., using nested loops) rather than actually implementing FL clients on real devices.

In this paper, we present *Flower*¹, a novel FL framework, that supports experimentation with both algorithmic and systems-related challenges in FL. Flower offers a stable, language and ML framework-agnostic implementation of the core components of a FL system, and provides higher-level abstractions to enable researchers to experiment and implement new ideas on top of a reliable stack. Moreover, Flower allows for rapid transition of existing ML training pipelines into a FL setup to evaluate their convergence properties and training time in a federated setting. Most importantly, Flower provides support for extending FL implementations to mobile and wireless clients, with heterogeneous compute, memory, and network resources.

As system-level challenges of limited compute, memory, and network bandwidth in mobile devices are not a major bottleneck for powerful cloud servers, Flower provides built-in tools to simulate many of these challenging conditions in a cloud environment and allows for a realistic evaluation of FL algorithms. Finally, Flower is designed with scalability in mind and enables research that leverages both a large number of connected clients and a large number of clients training concurrently. We believe that the capability to perform FL at scale will unlock new research opportunities as results obtained in small-scale experiments are not guaranteed to generalize well to large-scale problems. In summary, we make the following contributions to the ML systems literature:

- We present Flower, a novel FL framework that supports scalable algorithmic research and implementation of FL methods on edge devices and servers, including means to simulate real-world system conditions such as limited computational resources which are common for typical FL workloads.
- We describe the design principles and implementation details of Flower. In addition to being language- and ML framework-agnostic by design, Flower is also fully extendable and can incorporate emerging parameter averaging algorithms, new FL training strategies and communication protocols.

¹<https://flower.dev>

- Using Flower as the underlying framework, we present experiments that explore both algorithmic and system-level aspects of FL on five machine learning workloads with up to 1000 clients. Our results quantify the impact of various system bottlenecks such as client heterogeneity and fluctuating network speeds on FL performance.
- *Flower is open-sourced under Apache 2.0 License* and adopted by a variety of research projects focusing on FL-related questions. The community is welcome to participate in the development and contribute additional base-lines, functionality, or algorithms.

2 BACKGROUND AND RELATED WORK

FL builds on a vast body of prior work and has since been expanded in different directions. McMahan et al. (2017) introduced the basic federated averaging (FedAvg) algorithm and evaluated it in terms of communication efficiency. There is active work on privacy and robustness improvements for FL: A targeted model poisoning attack using Fashion-MNIST (Xiao et al., 2017) (along with possible mitigation strategies) was demonstrated by Bhagoji et al. (2018). Abadi et al. (2016b) propose an attempt to translate the idea of differential privacy to deep learning. Secure aggregation (Bonawitz et al., 2017) is a way to hide model updates from “honest but curious” attackers. Robustness and fault-tolerance improvements at the optimizer level are commonly studied and demonstrated, e.g., by Zeno (Xie et al., 2019a). Finally, there is an increasing emphasis on studying the performance of federated optimization techniques in heterogeneous data and system settings (Smith et al., 2017; Li et al., 2018; 2019).

The optimization of distributed training with and without federated concepts has been covered from many angles (Dean et al., 2012; Jia et al., 2018; Chahal et al., 2018; Sergeev & Balso, 2018; Dryden et al., 2016). Bonawitz et al. (2019) detail the system design of a large-scale Google-internal FL system. TFF (Google, 2020), PySyft (Ryffel et al., 2018), and LEAF (Caldas et al., 2018a) propose open source frameworks which are primarily used for simulations that run *homogeneous* clients on a single large machine. Flower unifies both perspectives by being open source and suitable for exploratory research, with scalability to expand into settings involving a large number of *heterogeneous* clients. Most of the mentioned approaches have in common that they implement their own systems to obtain the described results. The main intention of Flower is to provide a framework which would (a) allow to perform similar research using a common framework and (b) enable to run those experiments on a large number of devices.

3 FLOWER OVERVIEW

Flower attempts to bridge the gap between FL research and production. In this section we describe use cases that motivate our perspective, design goals, resulting framework architecture, and comparison to other frameworks.

3.1 Use Cases

We present some use cases for both researchers and engineers, and outline the role of Flower in their implementation.

Reproducible research. FL requires the implementation of several components (e.g., connection management on both client and server, FL loop, FL algorithm, client-side training, evaluation) before one can focus on the actual question at hand. Flower offers proven implementations of these components, thus enabling researchers to quickly experiment and implement new ideas on top of a reliable stack. Furthermore, having a library of existing federated optimization methods already implemented in Flower allows researchers to quickly compare their ideas against prior work, or base their experimentation on those works by modifying the provided implementations. A common platform for benchmarking FL approaches against each other is important because subtle implementation differences in the underlying distribution mechanisms can substantially harm the comparability of results.

Federating existing workloads. While FL has emerged as a promising technique, the breadth of ML tasks that have been translated to the federated setting remains quite narrow. Flower, by linking with existing ML frameworks and providing capabilities to run on both mobile devices and the cloud, will allow someone to take an existing ML training codebase and quickly create its federated equivalent.

Heterogeneous workloads. The norm of real-world FL will be heterogeneous devices collaborating on model training. Yet existing frameworks have very limited support for heterogeneity and focus on either server-side definition of client-side computations or ignore mobile clients completely by only offering cloud-based simulation of federated computations. Flower offers robust support for vastly different client environments collaborating in a single federation. It therefore allows to test the performance of existing algorithms in heterogeneous environments, but perhaps more importantly also enables research of algorithms which acknowledge and actively incorporate the heterogeneity assumption. Heterogeneity can stem from different sources, examples include connectivity, compute, hardware capabilities (such as different camera lenses affecting image quality), all potentially impacting FL convergence. Flower enables researchers to quantify the resulting effects on the learning process, both by running on diverse edge devices or by simulating certain constraints in the cloud.

Scaling research. Real-world FL setups often have a pool of hundreds or thousands of clients available for training. Yet many research configurations appear to be oversimplified in the sense that they only use, e.g., ten clients. Arguably this is due to the implementation effort of larger systems and lack of scalable FL frameworks available to researchers. A core driving ambition behind Flower is to enable research which leverages both a large number of connected clients and a large number of clients training concurrently. We hope this will encourage research that generalises better to the properties of real-world FL.

3.2 Design Goals

The creation of Flower was motivated by the observations that real-world FL workloads often face heterogeneous client environments, that the FL state-of-the-art advances quickly, that FL is difficult to implement, and that ML frameworks evolve rapidly. Based on those observations we define five major design goals for Flower:

- *ML framework-agnostic:* Given that ML frameworks are progressing rapidly and that large systems are likely contain devices running different frameworks, Flower should be compatible with a wide range of existing and future ML frameworks.
- *Client-agnostic:* Given heterogeneous environment on mobile clients, Flower should be interoperable with different programming languages, operating systems, and hardware settings.
- *Expandable:* Given the rate of change in FL, Flower should be expandable to enable both experimental research and adoption of recently proposed approaches.
- *Accessible:* Given the number and complexity of existing ML workloads, Flower should enable users to federate those pipelines with low engineering overhead.
- *Scalable:* Given that real-world FL would encounter a large number of clients, Flower should scale to a large number of concurrent clients to foster research on a realistic scale.

3.3 Core Framework Architecture

FL can be described as an interplay between global and local computations. Global computations are executed on the server side and responsible for orchestrating the learning process over a set of available clients. Local computations are executed on individual clients and have access to actual data used for training or evaluation of model parameters.

The architecture of the Flower core framework reflects that perspective and enables researchers to experiment with building blocks, both on the global and on the local level.

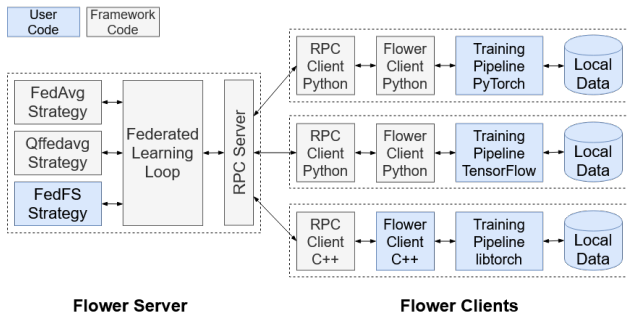


Figure 2. Flower core framework architecture.

Global logic for client selection, training configuration, parameter update aggregation, and federated or centralized model evaluation can be expressed through the *Strategy* abstraction. An implementation of the *Strategy* abstraction represents a single FL algorithm and Flower provides tested reference implementations of popular FL algorithms such as FedAvg (McMahan et al., 2017). Local logic on the other hand is mainly concerned with model training and evaluation on local data partitions. Flower acknowledges the breadth and diversity of existing ML pipelines and offers ML framework-agnostic ways to federate these, either on the protocol level or using the high-level *Client* abstraction. Figure 2 illustrates those components, §4 details the *Strategy* abstraction, and §5 client integration possibilities.

The Flower core framework implements the necessary infrastructure to run these workloads at scale. On the server side there are three major components involved: the FL loop, the RPC server, and a (user customizable) *Strategy*. Clients connect to the RPC server which is responsible for monitoring these connections and for sending and receiving Flower Protocol messages. The FL loops is at the heart of the FL process: it orchestrates the entire 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.

In summary, the FL loop asks the *Strategy* to configure the next round of FL, sends those configurations to the affected clients via the RPC server, receives the resulting client updates (or failures) from the clients via the RPC server, and delegates result aggregation to the *Strategy*. It takes the same approach for both federated training and federated evaluation, with the added capability of server-side evaluation (again, via the *Strategy*). The client side is (architecturally) simpler in the sense that it must only manage its own connection to the server and react to the messages received by calling user-provided training and evaluation functions.

A distinctive property of this architecture is that the server is unaware of the nature of connected clients, which allows

to train models across heterogeneous client platforms and implementations, including workloads comprised of different client-side ML frameworks. The framework manages underlying complexities such as connection handling, client life cycle, timeouts, and error handling for the researcher without prescribing a particular federation algorithm. The capability to perform FL at scale will unlock new research opportunities as results obtained in small-scale experiments often do not generalize well to large-scale problems.

3.4 Datasets and Baselines

The performance of FL algorithms is often influenced by the local datasets on each client – as such, in order to compare different FL algorithms in a fair and reproducible manner, it is important that they are implemented assuming the same data partitions across clients. Hence, with the goal of encouraging reproducible research, Flower provides a set of built-in datasets and partition functions that can be used to distribute the datasets across FL clients. Currently, Flower offers three datasets, namely Fashion-MNIST, CIFAR-10 and Speech Commands. While the first two datasets are commonly used for evaluating vision models, Speech Commands is a popular speech dataset used to train spoken keyword detection models. On top of these datasets, Flower has implemented partitioning functions which can split a dataset across clients in a user-defined way, e.g., {100% i.i.d}, {50% i.i.d., 50% non-i.i.d}. For example, in the {100% i.i.d} case, the partitions across clients follow the same distribution as the original dataset. However, in the {50% i.i.d., 50% non-i.i.d} setting, half of the data across each client is i.i.d, while the remaining half is sampled only from one of the available classes. Other types of datasets and partition functions can be added to Flower in the future. Flower also provides end-to-end FL baselines by combining Flower Datasets and Flower Strategies with popular model architectures for training client models.

3.5 FL Framework Comparison

We compare Flower to other FL toolkits, namely TFF (Google, 2020), PySyft (Ryffel et al., 2018), and LEAF (Caldas et al., 2018a). Table 1 provides an overview, with a more detailed description of those properties following thereafter.

- *Heterogeneous clients* refers to the ability to run workloads which include clients running on different platform using different languages, all in the same workload. FL targeting diverse edge devices will clearly have to assume pools of clients of many different types (e.g., phone, tablet, embedded). Flower supports such heterogeneous client pools through its language-independent client-side integration points. It is the only framework in our comparison that does so, with TFF and PySyft expecting a compatible client runtime, and LEAF being focused on Python-based

Table 1. Comparison of different FL frameworks.

	TFF	PySyft	LEAF	Flower
Heterogeneous clients				✓
Scalability	*	(✓)**		✓
Server-side definitions	✓	✓		
ML framework-agnostic		***		✓
Language-agnostic				✓
Baselines			✓	✓

planned * only Python-based instances **
limited to PyTorch and TF/Keras ***

simulations.

- *Scalability* is important to derive experimental results that generalize well. Single-machine simulation is limited as workloads including a large number of clients often exhibit vastly different properties. TFF and LEAF are, at the time of writing, constrained to single machine simulations. PySyft seems to be able to communicate over the network, but only to instances running Python. Flower allows workloads to scale to thousands of machines, including ones that are not able to run Python workloads out of the box (e.g., Android, iOS).
- *Server-side definition* of computations executed on the client describes a programming model which attempts to control the entire training process from a single point, the server. This approach is used by TFF and PySyft, which try to describe computations by taking a system-wide perspective. This approach can be advantageous for simulation, but it requires a full re-write of existing client-side ML pipelines. Flower, in contrast, treats global and local computations separately.
- *ML framework-agnostic* libraries allow researchers and users to leverage their previous investments in existing ML frameworks by providing universal integration points. This is a unique property of Flower: the ML framework landscape is evolving quickly and therefore the user should choose which framework to use for their local training pipelines. TFF is tightly coupled with TensorFlow, LEAF also has a dependency on TensorFlow, and PySyft provides hooks for PyTorch and Keras, but does not integrate with arbitrary tools.
- *Language-agnostic* describes the capability to implement clients in a variety of languages, a property especially important for research on mobile and emerging 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++ for resource constrained embedded devices. Flower achieves a fully language-agnostic interface by offering protocol-level integration. The other frameworks are based on

Python, with some of them indicating a plan to support Android and iOS in the future.

- *Baselines* allow the comparison of existing methods with new FL algorithms. Having existing implementations at ones disposal can greatly accelerate research progress. LEAF comes with a number of benchmarks built-in with different datasets. Flower currently implements a number of FL methods in the context of popular ML benchmarks, e.g., a federated training of CIFAR-10 (Krizhevsky et al., 2005) image classification, and has initial port of LEAF datasets such as FEMNIST and Shakespeare (Caldas et al., 2018b).

4 FEDERATION STRATEGIES

The *Strategy* abstraction is at the heart of the FL activity — it is basically synonymous with the FL algorithm performed. One design goal of Flower was to enable flexibility for both researchers to experiment with state-of-the-art approaches and application developers to tune the behaviour for their respective workload. The server achieves this flexibility through a plug-in architecture which delegates certain decisions to a user-provided implementation of the abstract base class *Strategy*. This strategy abstraction can therefore be used to inject arbitrary logic and customize core aspects of the FL process, e.g., client selection and update aggregation. As of now, five kinds of FL algorithms were successfully implemented (summarized in Table 2). These algorithms include plain FedAvg (McMahan et al., 2017), a fault-tolerant variant thereof, FedProx (Li et al., 2018) and Qffedavg (Li et al., 2019), but also an experimental method for extending FL to heterogeneous environments called FedFS. More details about FedFS are provided in the Appendix.

Researchers focusing on client-side questions are still able to run systems without having to implement the server-side details of FL algorithms. An implication of this design is that server strategies developed for one use case can easily be used with other use cases. This enables experimentation with new kinds of FL algorithms that generalize across tasks without having to re-implement core ideas for each task at question. It enables the creation of an open ecosystem: researchers can propose new strategies and offer them in stand-alone libraries, and application developers can compose the proposed strategies with the core framework and their individual workload.

5 CLIENT INTEGRATION

Local computations performed on the client face the challenge of diverse environments. Support for heterogeneous workloads requires Flower to offer integration capabilities for any of these environments. Other frameworks such as TFF attempt to address this by abstracting the client through

Table 2. Built-in federated learning algorithms available in Flower. New algorithms can be implemented using the *Strategy* interface.

Strategy	Description
FedAvg	Vanilla Federated Averaging (McMahan et al., 2017)
Fault Tolerant FedAvg	A variant of FedAvg that can tolerate faulty client conditions such as client disconnections or laggards.
FedProx	Implementation of the algorithm proposed by Li et al. (2018) to extend FL to heterogeneous network conditions.
Qffedavg	Implementation of the algorithm proposed by Li et al. (2019) to encourage fairness in FL.
FedFS	A new strategy for FL in scenarios of heterogeneous client computational capabilities. This strategy identifies fast and slow clients and intelligently schedules FL rounds across them to minimize the total convergence time.
FedOptim	A family of server-side optimizations that include FedAdagrad, FedYogi, and FedAdam as described in Reddi et al. (2021).

a runtime, which limits client integration capabilities by design. Flower instead opts to take a dual approach by offering both a low-level integration possibility via the Flower Protocol and a high-level convenience API for common types of environments.

5.1 Flower Protocol

The Flower Protocol is a low-level way of integrating workloads with Flower. It generally requires a client to handle well-defined message types and react appropriately, e.g., by receiving model parameters, optimizing them on local data, and then returning the updates parameters.

The Flower Protocol is comprised of two broad message categories: instructions and connectivity. *Instructions* are sent from the server to the client. The server might instruct the client to evaluate a given set of parameters on local data and expects the client to reply with the evaluation result. *Connectivity*-related messages can originate on both the client and the server: the server might decide that it will not select a particular client for some time and tell it to reconnect later, or the client might change its state such that it becomes necessary to disconnect from the server (e.g., a mobile client not being plugged in for charging any more).

Integrating directly with the Flower Protocol allows for sophisticated implementations, which can enable new kinds of FL methods. In addition, it allows for integration with experimental or emerging platforms that are not well supported by the larger ML ecosystem just yet, e.g., TensorFlow Lite Micro on the ESP32 (Dattu et al., 2020), and could therefore not be considered for FL research.

5.2 Client Abstraction

The Flower Protocol is broadly applicable, however it does not offer the ease-of-use Flower intends to provide. A high-level abstraction called *Client* is provided for common platforms and languages, built on top of the Flower Protocol. Its intent is to encapsulate functionality which is universal across workloads whilst enabling the user to focus on workload-specific logic. The *Client* abstraction calls user-provided code through an easy-to-implement interface whilst hiding underlying details such as connection management, Flower Protocol message types, and serialization.

One could consider the *Client* abstraction - and specialized versions thereof such as *KerasClient* - to be opinionated implementations of the Flower Protocol. They allow researchers to federate existing workloads with ease by only requiring them to write minimal amounts of glue code. An entire Keras-based workload was realized in less than 20 lines of Python code (appendix A.1). This work can span thousands of clients on different physical machines and demonstrates the usability of Flower given the underlying complexity of a distributed FL system.

6 IMPLEMENTATION

FL requires stable and efficient communication between clients and server. The Flower communication protocol is currently implemented on top of bi-directional gRPC (Foundation) streams. gRPC defines the types of messages exchanged and uses compilers to then generate efficient implementations for different languages such as Python, Java, or C++. A major reason for choosing gRPC was that it uses an efficient binary serialization format, which is especially important on low-bandwidth mobile connections. Bi-directional streaming allows for the exchange of multiple message without the overhead incurred by re-establishing a connection for every request/response pair.

Even though the current implementation uses gRPC, there is no inherent reliance on it. The internal Flower server architecture uses modular abstractions such that components that are not tied to gRPC are unaware of it. This could enable future versions of the server to support other RPC frameworks (e.g., Apache Avro), and even learn across heterogeneous clients with some connected through gRPC, and others through other RPC frameworks.

7 EVALUATION

We now evaluate Flower’s capabilities in supporting the research and implementation of real-world FL workloads. Our evaluation focuses on three aspects:

Scalability of Federated Learning. With the exception of a few production-grade systems (Bonawitz et al., 2019)

that are not open-source, prior research has evaluated FL algorithms on a small number of clients and small-scale datasets. In line with Flower’s aim of scaling FL research, we explore two questions:

- Can Flower help in studying the performance of federated optimization algorithms as they scale to a large number of clients?
- Can Flower support the federated training of models on web-scale datasets such as ImageNet, which can take several weeks to train?

Quantifying the system costs of FL. In practice, FL algorithms would run on battery-powered mobile and embedded clients with limited computational capabilities. Hence, it becomes crucial to quantify the latency and energy footprint of FL on these devices. We deploy Flower on Android smartphones and Nvidia embedded devices to evaluate the system costs associated with executing FL workloads.

Realism in Federated Learning. Much of the existing FL research is evaluated in simulated settings where the FL server and clients are run on the same machine, often using nested loops. Since Flower enables researchers to deploy FL algorithms in more realistic settings with computational and network heterogeneities across clients, can it lead to development of more robust federated optimization techniques?

The key results from our evaluation are:

- We demonstrate that Flower can scale to as many as 1000 FL clients with minimal overhead and help us uncover the properties of federated optimization algorithms at scale. To our knowledge, this is the largest FL experiment with an open-source framework, outside of simulations and closed-industrial platforms.
- Flower is a robust and stable framework that enables federating very large workloads such as ImageNet that take almost 15 days to train.
- Flower’s portability to different hardware, operating systems, and programming languages allows us to quantify the system costs associated with FL on real devices such as training time and energy consumption. Such quantification could be subsequently used to design new algorithms that trade-off between FL accuracy and system costs.
- Flower can throw light on the performance of FL under heterogeneous clients with different computational and network capabilities.

7.1 Scalability of Federated Learning

In this section, we evaluate Flower’s scalability.

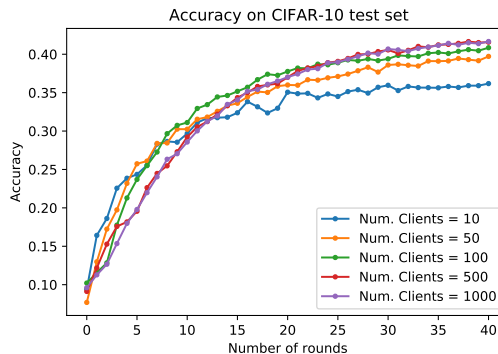


Figure 3. Accuracy on the CIFAR-10 test set after a given number of rounds. Increasing the number of clients improves the observed accuracy, however the gains saturate once we reach 100 clients.

7.1.1 Scaling FedAvg to 1000 distributed clients

We first evaluate Flower’s ability to perform federated training at scale. We do so by analyzing how the number of clients used during training affects the final accuracy.

Experiment Setup. We train a ResNet-18 model multiple times to correctly classify images from the CIFAR-10 dataset. We consider training scenarios with 10, 50, 100, 500, and 1000 clients, where each client has a local training set of 500 images. During each of the 40 rounds of FL, clients perform one epoch consisting of 32 local steps of SGD (batch size 16, momentum 0.9) before sending the updated model parameters back to the server. The server samples 10% of all connected clients to participate in each round.

To properly scale our experiment to these conditions and to simulate the fact that different clients will have different data, we have augmented the original CIFAR-10 training set from 50k to 500k samples using standard image augmentation techniques such as random rotations, horizontal and vertical flips, and color jitters such as contrast, brightness, saturation and hue adjustments. We call this the Extended CIFAR-10 (EC-10) dataset. Finally, we partition EC-10 into 1000 IID subsets (500 samples each) and distribute one subset to each client. The FL server and all the clients are hosted on virtual machines in a cloud platform.

Results. We report our results in Figure 3, which shows how accuracy on the CIFAR-10 test set improves with the number of rounds and the number clients in the system. Since clients will have the same number of images in all setups, increasing the number of clients also means increasing the amount of data available to the training system. However, we observe that the increase in accuracy caused by the increase in the number of clients quickly saturates for more than 50 clients. Moreover, scaling from 500 to 1000 clients has no performance gains for FL. We can attribute this saturation to the simplicity of the dataset, which has only ten classes and relatively small input size.

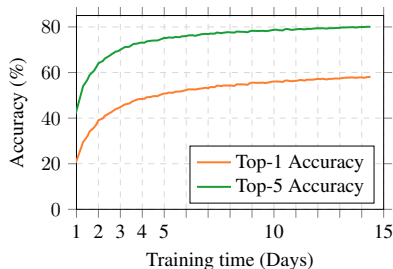


Figure 4. Training time reported in days and accuracies (Top-1 and Top-5) for an ImageNet federated training with Flower.

This experiment with Flower throws light on an important question in FL: can federated optimization algorithms achieve the same levels of accuracy as centralized training? Our results show that even at a scale of 1000 clients, FedAvg cannot achieve the same accuracy as centralized training on EC-10 (around 70.1%). This suggests that researchers need to design better federated algorithms for this dataset, rather than scale existing algorithms to more clients.

7.1.2 Scaling FedAvg to ImageNet-scale datasets

We now demonstrate that Flower can not only scale to a large number of clients, but it can also support training of FL models on web-scale workloads such as ImageNet. To the best of our knowledge, this is the first-ever attempt at training ImageNet in a FL setting.

Experiment Setup. We use the ILSVRC-2012 ImageNet partitioning (Russakovsky et al., 2015) that contains 1.2M pictures for training and a subset composed of 50K images for testing. We train a ResNet-18 model on this dataset in a federated setting with 50 clients equipped with four physical CPU cores. To this end, we partition the ImageNet training set into 50 IID partitions and distribute them on each client. During training, we also consider a simple image augmentation scheme based on random horizontal flipping and cropping.

Results. Figure 4 shows the results on the test set of ImageNet obtained by training a ResNet-18 model. It is worth to mention that based on 50 clients and 3 local epochs, the training lasted for about 15 days demonstrating Flower’s potential to run long-term and realistic experiments.

We measured top-1 and top-5 accuracies of 59.1% and 80.4% respectively obtained with FL compared to 63% and 84% for centralised training. First, it is clear from Figure 4 that FL accuracies could have increased a bit further at the cost of a longer training time, certainly reducing the gap with centralised training. Then, the ResNet-18 architecture relies heavily on batch-normalisation, and it is unclear how the internal statistics of this technique behave in the context of FL, potentially harming the final results. As expected, the scalability of Flower helps with raising and investing new issues related to federated learning.

Table 3. Flower clients on 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

For such long-term experiments, one major risk is that client devices may go offline during training, thereby nullifying the training progress. Flower’s built-in support for keeping the model states on the server and resuming the federated training from the last saved state in the case of failures came handy for this experiment.

7.2 Quantifying the System Costs of FL

Flower can assist researchers in quantifying the system costs associated with running FL on real devices, and designing algorithms that trade-off between system costs and FL accuracy. In this section, we present the results of deploying Flower on Android devices in the Amazon AWS Device Farm and on Nvidia Jetson TX2 edge accelerator.

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. More details about them are provided in the Appendix.

Experiment Setup. We run the Flower server configured with the FedAvg strategy and host it on a cloud virtual machine. Nvidia TX2 edge devices support full-fledged PyTorch as the ML framework – this means we could successfully port existing PyTorch training pipelines to implement FL clients on them. On the other hand, smartphones running the Android OS currently do not have extensive on-device training support with TensorFlow or PyTorch. To counter this issue, we leverage TensorFlow Lite to implement Flower clients on Android smartphones. While TFLite is primarily designed for on-device inference, we exploit its capabilities to do on-device model personalization to implement a FL client application. More specifically, we use a pre-trained and frozen MobileNetV2 base model for extracting image features and train a 2-layer DNN (using FL) as the classifier that operates on the extracted features.

Finally, to scale our experiments to a reasonably large number of mobile clients with different OS versions, we deploy Flower on the Amazon AWS Device Farm (AWS, 2020) that enables deploying applications on real mobile devices accessed through AWS. Table 3 list the mobile devices from AWS Device Farm used in our evaluation.

Results. In Table 4, we present various performance metrics obtained on Nvidia TX2 and the Android devices. First, we train a ResNet-18 model on the CIFAR-10 dataset on 10

Table 4. 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 clients as we vary the number of local epochs. We use the CIFAR10 dataset and train a ResNet-18 model on it. 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.

Table 5. Average time and total power consumption (on-device training+communication+aggregation) per round for different devices with different power configurations.

Device	Time per Epoch(s)	Energy (kJ)
Xavier-NX (10W, 4 cores)	203.0	5.6
Jetson-TX2 (7.5W, 4 cores)	382.2	7.6
Xavier-NX (15W, 6 cores)	173.0	7.4
Jetson-TX2 (15W, 6 cores)	271.4	11.1

Nvidia TX2 clients. In Table 4a, 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. Table 5 shows the trade-off between energy consumption and training time for devices that allow for different power profiles.

While the accuracy metrics in Table 4a 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 enabled by Flower. 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 of FL.

Next, we train a 2-layer DNN classifier on top of a pre-trained MobileNetV2 model on Android clients for the Office-31 dataset. In Table 4b, 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

Table 6. 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	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)

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.

7.3 Realism in Federated Learning

Flower facilitates the deployment of FL on real-world devices. While this property is beneficial for production-grade systems, can it also assist researchers in developing better federated optimization algorithms? In this section, we study two realistic scenarios of FL deployment.

Computational Heterogeneity across Clients. In real-world, FL clients will have vastly different computational capabilities. While newer smartphones are now equipped with mobile GPUs, other phones or wearable devices may have a much less powerful processor. How does this computational heterogeneity impact FL?

For this experiment, we use a Nvidia TX2 as the client device, which has one Pascal GPU and six CPU cores. The results on training time presented earlier in Table 4a were obtained when the ResNet-18 model is trained on the GPU. In Table 6, we show that if the same model is trained on the CPU with local epochs $E = 10$, it would take $1.27\times$ more time to obtain the same accuracy as the GPU training.

Once we obtain this quantification of computational heterogeneity using Flower, we can design better federated optimization algorithms. As an example, we implemented 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 a FL round on the TX2 GPU. If we set the same time as a cutoff for CPU training ($\tau = 1.99$ mins) as shown in Table 6, we can 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.

Heterogeneity in Network Speeds. An important consideration for any FL system is to choose a set of participating clients in each training round. In the real-world, clients are distributed across the world and vary in their download and upload speeds. Hence, it is critical for any FL system to study how client selection can impact the overall FL training time. We now present an experiment with 40 clients collaborating to train a 4-layer deep CNN model for the FashionMNIST dataset. More details about the dataset and network architecture are presented in the Appendix.

Using Flower, we instantiate 40 clients on a cloud platform and fix the download and upload speeds for each client using the WONDERSHAPER library. Each client is representative of a country and its download and upload speed is set based on a recent market survey of 4G and 5G speeds in different countries (OpenSignal, 2020).

The x-axis of Figure 5 shows countries arranged in descending order of their network speeds: country indices 1-20 represent the top 20 countries based on their network speeds (mean download speed = 40.1Mbps), and indices 21-40 are the bottom 20 countries (mean download speed = 6.76Mbps). We observe that if all clients have the network speeds corresponding to Country 1 (Canada), the FL training finishes in 8.9 mins. As we include slower clients in FL, the training time gradually increases, with a major jump around index = 17. On the other extreme, for client speeds corresponding to Country 40 (Iraq), the FL training takes 108 minutes.

There are two key takeaways from this experiment: a) Using Flower, we can profile the training time of any FL algorithm under scenarios of network heterogeneity, b) we can leverage these insights to design sophisticated client sampling techniques. For example, during subsequent rounds of federated learning, we could monitor the number of samples each client was able to process during a given time window and increase the selection probability of slow clients to balance the contributions of fast and slow clients to the global model. The FedFS strategy introduced in Table 2 and detailed in appendix A.2 works on this general idea, and reduces the convergence time of FL by up to 30% over vanilla FedAvg which randomly samples clients in each round.

8 LIMITATIONS AND FUTURE WORK

In the following, we highlight limitations of the current Flower design and evaluation – along with discussing future areas of research.

Limitations. Libraries for efficient training on mobile devices are still in a nascent stage. But for any FL solution it is clearly a critical ingredient. By design, Flower can

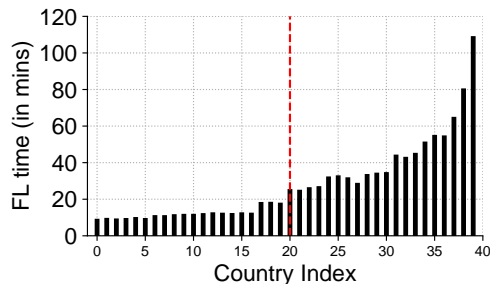


Figure 5. Effect of network heterogeneity in clients on FL training time. Using this quantification, we designed a new client sampling strategy called FedFS (detailed in the Appendix).

leverage any ML framework (e.g., TensorFlow or PyTorch) which maximizes its ability to use existing training pipelines. However, it also means Flower inherits the limitations of these frameworks that currently offer very limited support for on-device training (unlike the extensive solutions for on-device inference). It is anticipated that ML frameworks will address this short-coming within the next 12 months. Furthermore, Flower already includes expanded on-device training via the integration of TFLite model personalization routines (described in §7.2) which can be treated as a proof-of-concept for future support. Such restrictions may prevent certain models out-of-the-box from being easily deployed, however because of the rich set of APIs (e.g., Java/Swift) it still remains possible for Flower users to implement solutions that circumvent such barriers.

Looking Ahead. The most exciting and timely next step for Flower will be to examine a variety of FL algorithms and results at a much large *scale* and a much great level of *heterogeneity* than has previously been possible. To date FL approaches in the literature are rarely evaluated with large numbers of client participants (such as in the thousands) – and virtually are never tested under a pool of different mobile devices, as would be the norm for FL systems targeting mobile platforms. In future work, we plan to use Flower to revisit a number of key FL results and test if these results hold up under more realistic conditions. Performing such experiments become much more feasible under Flower – and we anticipate this will uncover many situations existing FL solutions do not perform as expected.

Complementing conventional supervised applications, we also expect Flower to be indispensable in the exploration of the rapidly maturing area of unsupervised, semi-supervised and self-learning (Xie et al., 2019b). FL using supervised methods are often not practical simply because it is difficult to acquire labeled data from users. But in contrast, devices have plentiful access to virtually unlimited amounts of unlabeled data. Furthermore, these learning approaches significantly increase the amount of data to be trained upon as unlabeled data is much more prevalent and so benefits from FL ability to distribute the training computation.

9 CONCLUSION

We have presented Flower – a novel framework that is specifically designed to advance FL research by offering a new way to run fully heterogeneous FL workloads at scale. Although Flower is broadly useful across a range of FL settings, we believe that it will be a true *game-changer* for reducing the disparity between FL research and production environments. Through the provided abstractions and components, researchers can federated existing ML workloads - regardless of the ML framework used - in as little as 20 lines of Python code. We further evaluate the capabilities of Flower in experiments that target both scale and systems heterogeneity by scaling FL up to 1000 clients, performing the first federated training on ImageNet (to the best of our knowledge), measuring FL energy consumption on a cluster of Nvidia Jetson TX2 devices, optimizing convergence time under limited bandwidth, and illustrating a deployment of Flower on a range of Android mobile devices in the AWS Device Farm. Flower is open-sourced under Apache 2.0 License and we look forward to more community contributions to it.

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, 2016a. URL <https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf>.
- Abadi, M., Chu, A., Goodfellow, I., McMahan, B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *23rd ACM Conference on Computer and Communications Security (ACM CCS)*, pp. 308–318, 2016b. URL <https://arxiv.org/abs/1607.00133>.
- AWS. Aws device farm. <https://aws.amazon.com/device-farm/>, 2020. accessed 25-Mar-20.
- Bhagoji, A. N., Chakraborty, S., Mittal, P., and Calo, S. B. Analyzing federated learning through an adversarial lens. *CoRR*, abs/1811.12470, 2018. URL <http://arxiv.org/abs/1811.12470>.
- Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, pp. 1175–1191, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4946-8. doi: 10.1145/3133956.3133982. URL <http://doi.acm.org/10.1145/3133956.3133982>.
- 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. In *SysML 2019*, 2019. URL <https://arxiv.org/abs/1902.01046>. To appear.
- 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, 2018a. URL <http://arxiv.org/abs/1812.01097>.
- Caldas, S., Wu, P., Li, T., Konečný, J., McMahan, H. B., Smith, V., and Talwalkar, A. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018b. URL <https://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>.
- Chollet, F. et al. Keras. <https://github.com/fchollet/keras>, 2015.
- 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>.
- Dattu, V., Patwardhan, A., and Sovani, K. Announcing tensorflow lite micro support on the esp32. <https://blog.tensorflow.org/2020/08/announcing-tensorflow-lite-micro-esp32.html>, 2020. accessed 08-Oct-20.
- Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, pp. 1223–1231, USA, 2012. Curran Associates Inc. URL <http://dl.acm.org/citation.cfm?id=2999134.2999271>.
- Dryden, N., Jacobs, S. A., Moon, T., and Van Essen, B. Communication quantization for data-parallel training of deep neural networks. In *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments, MLHPC '16*, pp. 1–8, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-5090-3882-4. doi: 10.1109/MLHPC.2016.4. URL <https://doi.org/10.1109/MLHPC.2016.4>.

- Foundation, C. N. C. grpc: A high performance, open-source universal rpc framework. URL <https://grpc.io>. Accessed: 2020-03-25.
- 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.
- Georgiev, P., Lane, N. D., Mascolo, C., and Chu, D. Accelerating mobile audio sensing algorithms through on-chip GPU offloading. In Choudhury, T., Ko, S. Y., Campbell, A., and Ganesan, D. (eds.), *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'17, Niagara Falls, NY, USA, June 19-23, 2017*, pp. 306–318. ACM, 2017. doi: 10.1145/3081333.3081358. URL <https://doi.org/10.1145/3081333.3081358>.
- Google. Tensorflow federated: Machine learning on decentralized data. <https://www.tensorflow.org/federated>, 2020. accessed 25-Mar-20.
- Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. Federated learning for mobile keyboard prediction, 2019.
- 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>.
- Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). *Online*, 2005. URL <http://www.cs.toronto.edu/~kriz/cifar.html>.
- 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>.
- 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.
- Li, T., Sanjabi, M., and Smith, V. Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497*, 2019.
- LiKamWa, R., Hou, Y., Gao, J., Polansky, M., and Zhong, L. Redeye: Analog convnet image sensor architecture for continuous mobile vision. In *Proceedings of the 43rd International Symposium on Computer Architecture, ISCA '16*, pp. 255–266. IEEE Press, 2016. ISBN 9781467389471. doi: 10.1109/ISCA.2016.31. URL <https://doi.org/10.1109/ISCA.2016.31>.
- 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.
- OpenSignal. The state of mobile network experience 2020: One year into the 5g era. <https://www.opensignal.com/reports/2020/05/global-state-of-the-mobile-network>, 2020. accessed 10-Oct-20.
- Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., and McMahan, H. B. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=LkFG31B13U5>.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.
- 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>.
- Sergeev, A. and Balso, M. D. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018. URL <http://arxiv.org/abs/1802.05799>.

- Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pp. 4424–4434, 2017.
- Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- Xie, C., Koyejo, S., and Gupta, I. Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6893–6901, Long Beach, California, USA, 09–15 Jun 2019a. PMLR. URL <http://proceedings.mlr.press/v97/xie19b.html>.
- Xie, Q., Luong, M.-T., Hovy, E., and Le, Q. V. Self-training with noisy student improves imagenet classification, 2019b.
- Yao, Y., Li, H., Zheng, H., and Zhao, B. Y. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19*, pp. 2041–2055, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367479. doi: 10.1145/3319535.3354209. URL <https://doi.org/10.1145/3319535.3354209>.

A APPENDIX

A.1 Flower Code Example

Flower code example showcasing a (simplified) version of federated CIFAR-10 (Krizhevsky et al., 2005) image classification implemented in Keras (Chollet et al., 2015).

```

1 import tensorflow as tf
2 from tensorflow.keras.datasets import cifar10
3 from tensorflow.keras.applications import MobileNetV2
4 import flwr as fl
5 from flwr.client.keras_client import KerasClient
6
7 # Load and compile Keras model
8 model = MobileNetV2((32, 32, 3), classes=10, weights=
9     None)
10 model.compile("adam", "sparse_categorical_crossentropy"
11     , metrics=["accuracy"])
12
13 # Load CIFAR-10 dataset
14 (x_train, y_train), (x_test, y_test) = cifar10.
15     load_data()
16
17 # Define Flower client
18 class CifarClient(KerasClient):
19     def get_weights(self):
20         return model.get_weights()
21
22     def fit(self, weights, config):
23         model.set_weights(weights)
24         model.fit(x_train, y_train)
25         return model.get_weights(), len(x_train), len(
26             x_train)
27
28     def evaluate(self, weights, config):
29         model.set_weights(weights)
30         loss, accuracy = model.evaluate(x_test, y_test)
31         return len(x_test), loss, accuracy
32
33 # Start Flower client
34 fl.client.start_keras_client(
35     server_address="[:]:8080", client=CifarClient())
    
```

Listing 1. CIFAR-10 Image Classification (client.py)

```

1 import flwr as fl
2
3 # Start Flower server for three rounds of FL
4 fl.server.start_server( config={"num_rounds": 3})
    
```

Listing 2. CIFAR-10 Image Classification (server.py)

A.2 FedFS Algorithm

We introduce *Federating: Fast and Slow (FedFS)* to overcome the challenges arising from heterogeneous devices and non-IID data. *FedFS* acknowledges the difference in compute capabilities inherent in networks of mobile devices by combining partial work, importance sampling, and dynamic timeouts to enable clients to contribute equally to the global model.

Partial work. Given a (local) data set of size m_k on client k , a batch size of B , and the number of local training epochs E , FedAvg performs $E \frac{m_k}{B}$ (local) gradient updates $\theta^k \leftarrow \theta^k - \eta \nabla \ell(b; \theta^k)$ before returning θ^k to the server. The asynchronous setting treats the success of local update computation as binary. If a client succeeds in computing $E \frac{m_k}{B}$ mini-batch updates before reaching a timeout Δ , their

weight update is considered by the server, otherwise it is discarded. The server then averages all successful $\theta_{k \in \{0, \dots, K\}}$ updates, weighted by m_k , the number of training examples on client k .

This is wasteful because a clients' computation might be discarded upon reaching Δ even if it was close to computing the full $E \frac{m_k}{B}$ gradient updates. We therefore apply the concept of partial work (Li et al., 2018) in which a client submits their locally updated θ_k upon reaching Δ along with c_k , the number of examples actually involved in computing θ_k , even if $c_k < E \frac{m_k}{B}$. The server averages by c_k , not m_k , because c_k can vary over different rounds and devices depending on a number of factors (device speed, concurrent processes, Δ , m_k , etc.).

Intuitively, this leads to more graceful performance degradation with smaller values for Δ . Even if Δ is set to an adversarial value just below the completion time of the fastest client, which would cause *FedAvg* to not consider any update and hence prevent convergence, *FedFS* would still progress by combining K partial updates. More importantly it allows devices which regularly discard their updates because of lacking compute capabilities to have their updates represented in the global model, which would otherwise overfit the data distribution on the subset of faster devices in the population.

Importance sampling. Partial work enables *FedFS* to leverage the observed values for c_k^r (with $r \in \{1, \dots, t\}$, the amount of work done by client k during all previous rounds up to the current round t) and $E^r m_k$ (with $r \in \{1, \dots, t\}$, the amount of work client k was maximally allowed to do during those rounds) for client selection during round $t + 1$. c and m can be measured in different ways depending on the use case. In vision, c_k^t could capture the number of image examples processed, whereas in speech c_k^t could measure the accumulated duration of all audio samples used for training on client k during round t . $c_k^t < E^t m_k$ suggests that client k was not able to compute $E^t \frac{m_k}{B}$ gradient updates within Δ_t , so its weight update θ_k^t has less of an impact on the global model θ compared to an update from client j with $c_j^t = E^t m_j$. *FedFS* uses importance sampling for client selection to mitigate the effects introduced by this difference in client capabilities. We define the work contribution w_k of client k as the ratio between the actual work done during previous rounds $c_k = \sum_{r=1}^t c_k^r$ and the maximum work possible $\hat{c}_k = \sum_{r=1}^t E^r m_k$. Clients which have never been selected before (and hence have no contribution history) have $w_k = 0$. We then sample clients on the selection probability $1 - w_k + \epsilon$ (normalized over all $k \in \{1, \dots, K\}$), with ϵ being the minimum client selection probability. ϵ is an important hyper-parameter that prevents clients with $c_k^t = E^t m_k$ to be excluded from future rounds. Basing the client selection probability on a clients' previous contribu-

Algorithm 1: FedFS

```

begin Server  $T, C, K, \epsilon, r_f, r_s, \Delta_{max}, E, B,$ 
    initialise  $\theta_0$ 
    for round  $t \leftarrow 0, \dots, T - 1$  do
         $j \leftarrow \max(\lfloor C \cdot K \rfloor, 1)$ 
         $\mathcal{S}_t \leftarrow$  (sample  $j$  distinct indices from  $\{1, \dots, K\}$ 
            with  $1 - w_k + \epsilon$ )
        if fast round ( $r_f, r_s$ ) then
             $\Delta_t = \Delta^f$ 
        else
             $\Delta_t = \Delta^s$ 
        end
        for  $k \in \mathcal{S}_t$  do in parallel
             $\theta_{t+1}^k, c_k, m_k \leftarrow$  ClientTraining( $k, \Delta_t, \theta_t, E, B,$ 
                 $\Delta_t$ )
        end
         $c_r \leftarrow \sum_{k \in \mathcal{S}_t} c_k$ 
         $\theta_{t+1} \leftarrow \sum_{k \in \mathcal{S}_t} \frac{c_k}{c_r} \theta_{t+1}^k$ 
    end
end
    
```

tions (w_k) allows clients which had low contributions in previous rounds to be selected more frequently, and hence contribute additional updates to the global model. Synchronous *FedAvg* is a special case of *FedFS*: if all clients are able to compute $c_k^t = E^t m_k$ every round, then there will be no difference in w_k and *FedFS* samples amongst all clients with a uniform client selection probability of $\frac{1}{k}$.

Alternating timeout. Gradual failure for clients which are not able to compute $E^t \frac{m_k}{B}$ gradient updates within Δ_t and client selection based on previous contributions allow *FedFS* to use more aggressive values for Δ . One strategy is to use an alternating schedule for Δ in which we perform r_f “fast” rounds with small Δ^f and r_s “slow” rounds with larger Δ^s . This allows *FedFS* to be configured for either improved convergence in terms of wall-clock time or better overall performance (e.g., in terms for classification accuracy).

FedFS algorithm. The full *FedFS* algorithm is given in 1.

A.3 Datasets and Network Architectures

We use the following datasets and network architectures for our experiments.

CIFAR-10 consists of 60,000 images from 10 different object classes. The images are 32 x 32 pixels in size and in RGB format. We use the training and test splits provided by the dataset authors — 50,000 images are used as training data and remaining 10,000 images are reserved for testing. For the experiment with 1000 clients, we have augmented the CIFAR-10 training set from 50k to 500k samples using standard image augmentation techniques such as random rotations, horizontal and vertical flips, and color jitters such as contrast, brightness, saturation and hue adjustments. We call

this the Extended CIFAR-10 (EC-10) dataset. The ResNet-18 architecture is used for federated training of CIFAR-10.

Office-31 contains images of common office objects (e.g., printers, tables) belonging to 31 different classes. The dataset has images captured from different sources (a web camera, a DSLR camera and Amazon product images) — for our experiments, we only use the object images from ‘Amazon’ as the FL workload primarily because the images from other sources are fewer in number. In total, we use 2900 images (10% are held out for testing) which are 300 x 300 pixels in size and in RGB format. We use a pre-trained and frozen MobileNetV2 base model for extracting image features and train a 2-layer DNN (using FL) as the classifier that operates on the extracted features.

Fashion-MNIST consists of images of fashion items (60,000 training, 10,000 test) with 10 classes such as trousers or pullovers. The images are 28 x 28 pixels in size and in grayscale format. We use a 2-layer CNN followed by 2 fully-connected layers for training a model on this dataset.

ImageNet. We use the ILSVRC-2012 ImageNet partitioning (Russakovsky et al., 2015) that contains 1.2M pictures for training and a subset composed of 50K images for testing. The ResNet-18 architecture is used for federated training of ImageNet.

A.4 Survey of FL papers with respect to the number of clients.

Table 7. Surveying the scale in terms of the number of clients used in FL research.

Paper	No. of total clients (Max)
How To Backdoor Federated Learning	100
Federated Learning via Over-the-Air Computation	20
Local Model Poisoning Attacks to Byzantine-Robust Federated Learning	100
Federated Learning over Wireless Fading Channels	30
Clustered Federated Learning: Model-Agnostic Distributed Multi-Task Optimization under Privacy Constraints	20
Personalized Federated Learning: A Meta-Learning Approach	50
Incentive Design for Efficient Federated Learning in Mobile Networks: A Contract Theory Approach	100
Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data	10
Client-Edge-Cloud Hierarchical Federated Learning	50
Device Scheduling with Fast Convergence for Wireless Federated Learning	20
BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning	9
Energy-Aware Analog Aggregation for Federated Learning with Redundant Data	50
Mix2FLD: Downlink Federated Learning After Uplink Federated Distillation With Two-Way Mixup	10
Three Approaches for Personalization with Applications to Federated Learning	100
A Fairness-aware Incentive Scheme for Federated Learning	100
Federated Learning With Quantized Global Model Updates	40
On Safeguarding Privacy and Security in the Framework of Federated Learning	50
Federated Learning With Differential Privacy: Algorithms and Performance Analysis	100
Fast-Convergent Federated Learning	1000
Dispersed Federated Learning: Vision, Taxonomy, and Future Directions	54
Federated Learning With Cooperating Devices: A Consensus Approach for Massive IoT Networks	80
Federated Learning with Matched Averaging	66
Communication Efficient Federated Learning over Multiple Access Channels	2
Harnessing Wireless Channels for Scalable and Privacy-Preserving Federated Learning	100
Age-Based Scheduling Policy for Federated Learning in Mobile Edge Networks	100
Convergence Time Optimization for Federated Learning over Wireless Networks	15
Salvaging Federated Learning by Local Adaptation	100
Multi-Armed Bandit Based Client Scheduling for Federated Learning	20
Wireless Federated Learning with Local Differential Privacy	30
FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare	30
Convergence of Update Aware Device Scheduling for Federated Learning at the Wireless Edge	40
Byzantine-resilient Secure Federated Learning	40
FetchSGD: Communication-Efficient Federated Learning with Sketching	50000
FLet: Online Federated Learning via Staleness Awareness and Performance Prediction	40
Federated Optimization in Heterogeneous Networks	1000
LoAdaBoost: loss-based AdaBoost federated machine learning with reduced computational complexity on IID and non-IID intensive care data	90
Astraea: Self-balancing Federated Learning for Improving Classification Accuracy of Mobile Deep Learning Applications	500
Federated Learning with Non-IID Data	10
Differentially Private AirComp Federated Learning with Power Adaptation Harnessing Receiver Noise	100