UNIVERSITY OF CAMBRIDGE

Re-thinking Federated Learning

PhD Proposal

Lorenzo Sani

Jesus

First year report submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Contents

Chapter 1

Introduction

In recent years, technologies related to Artificial Intelligence (AI) have been undergoing a profound review from the privacy perspective. Since many of the applications nowadays are meant to address the needs of single users, such as personalised recommendations, it is crucial to use personal data to train models, which comes with privacy concerns. On the one hand, the data is the backbone on which we train AI algorithms, and the more data available, the better the algorithms perform. On the other hand, the data is often sensitive and can be used to infer private information about individuals. Regulators and the public have raised such concerns regarding AI applications. The research community has also been actively investigating the privacy implications of AI algorithms and developing privacy-preserving AI techniques. On this side, Federated Learning (FL) has emerged as a promising approach to training Machine Learning (ML) models in a decentralised environment in which data is distributed across multiple devices, which are usually referred to as *clients*. In its most straightforward fashion, FL is a distributed technique in which a central server orchestrates the training of an ML model by iteratively sending the current model parameters to the clients, which compute the gradients of the loss function on their local data and send them back for aggregation. Using FL, data is *never* shared between the federation participants. The growing interest in FL is projecting the research community and the industry into a new era of privacy-preserving AI.

Since FL was first proposed in 2017, the community has always limited FL characterisation as a privacy-preserving framework for distributed training of ML models. However, recent advancements in the topic have demonstrated that FL opens another great opportunity: to train ML models on enormous distributed datasets. FL possibly enables training ML models on data stored on a massive population of edge devices, such as smartphones, allowing an unprecedentedly broad basin of information to learn. The above sustains the thesis that the privacy-preserving properties of FL will enable the training of ML models on data that would otherwise be inaccessible due to privacy concerns. The experience taught us that such algorithms' robustness and performance strongly depend on the data available for training. Usually, the more data is available, the more these technologies are reliable in modelling the real world, thus providing users with better predictions. However, the current state of the art is still far from defining a standard framework for dealing with *large-scale federated learning* settings for diverse reasons. On top of these is the topic's relative novelty. In addition, the lack of standard definitions in the realm of FL regarding the terminology and the problem statement has led to a proliferation of different approaches and solutions, which are often not directly comparable. Finally, the open challenges of such a system, such as the heterogeneity of the clients, the communication and computation constraints, and additional privacy concerns, have not been addressed entirely. With this proposal, I aim to plan a research path to address some of these challenges and to investigate the opportunities that *large-scale FL* offers, which we define as an FL setting having one or many elements being a vast amount, e.g., the number of clients, the number of total samples, the number of parameters composing the global model.

Recently, the research community has identified some clues that suggest that the FedAvg algorithm, or more generally those algorithms based on weighted averaging procedure, such as FedProx, (say many more), might not be the best choice for training *large-scale FL* models. The most relevant observation comes from analysing the clients' updates received by the server. These often conflict with each other since they partially point in opposite directions. The weighted averaging procedure of FedAvg may cancel out helpful information, while more advanced aggregation strategies only partially mitigate this behaviour by constraining the updates with regularisation terms. In the best case, the model can partially recover the wasted information during the following training rounds but not at all in the worst case. The above results in a suboptimal training procedure, requiring more communication rounds to converge to a reasonable solution or ending in an underperforming model. In *large-scale FL*, we want to leverage every precious piece of information represented by the massive amount of data from the clients available to enable clients to benefit from the federation. This will require us to investigate aggregation techniques beyond the averaging procedure.

In this context, the famous large models (LM) represent a potential opportunity for analysing how clients' updates interact with the aggregation procedure. Since LMs come with a novel training framework composed of pre-training and fine-tuning, we can leverage the pre-trained model's knowledge as a frame of reference for such analysis. However promising, this approach is not without challenges. Many state-of-the-art models have a massive amount of parameters. Thus, they are not suitable for running on clients that are resource-constrained in most cases. It has been years since the research community has investigated compression techniques such as pruning, sparsification, and quantisation to solve this issue. However, it is still unclear how to apply them in the context of FL. Recent developments in this area have demonstrated, in the centralised setting, the existence of the so-called *winning tickets*, which are sparse subnetworks that can achieve the same accuracy as the original model. Such techniques exploiting training data features to achieve some degree of compression are promising in the context of FL. On the one hand, we can take advantage of the frame of reference provided by large pre-trained models. On the other hand, we can leverage compression techniques to reduce the dimensionality of the clients' update space to enable a more manageable analysis of their interaction with the aggregation procedure. This context can set the ground for a new generation of algorithms the community will use to train efficiently in *large-scale FL* settings.

Chapter 2

Background and Related Work

The discussion in this chapter aims to describe the relevant work that has been conducted on the topics highlighted in the introduction of this report. Since the area to cover is broad, this chapter will focus on those works directly impacting the research being carried out. The chapter is structured as follows: Section 2.1 presents FL with the state-of-the-art method available and a particular focus on large-scale settings; Section 2.2 delineate the work at the intersection of GNNs and FL focussing on GNN-driven aggregation; Section 2.3 shows the relevant literature that leverages compression techniques in FL; Section 2.4 concludes the literature review by presenting open-source FL frameworks with their open problems.

2.1 Federated Learning

Federated Learning (FL) has been the subject of a comprehensive set of studies that have exponentially increased in numbers since [53] first introduced it. FL has become very popular since it is a promising solution to the problem of training machine learning models on data distributed across multiple devices. In particular, it allows private data to be kept (protected) on the devices which have collected it, thus relieving producers from developing a complex framework for respecting data privacy regulations such as GDPR [62] or CCPA [62]. In its most general fashion, FL is a machine learning paradigm composed of a server orchestrating distributed training on a federation of clients by iteratively interleaving local training of the ML model on clients and central aggregation of the updates. The server is responsible for initialising the global model and aggregating the updates of the clients. The following scheme depicts the iterative procedure of FL:

1. The server sends the current global model to a subset of selected clients.

2. The clients train the model on their local data and send their updates to the server.

3. The server aggregates the updates and updates the global model.

The FL training comprises iterations, called *rounds*, of these steps. As pointed out in [31], the research community still has to address many open problems and challenges for making FL a standard framework for training machine learning models in a private scenario. The most notable of which are the following: improving the performance of the algorithms in terms of efficiency and effectiveness; defining strict security and privacy guarantees along with a solid framework for assessing them; dealing with failing actors in the federation, which the literature often treats on par with malicious attacks since their commonalities; challenging the issues concerning fairness and bias in the data; and finally, investigating the impact of real devices in such a decentralised framework to the system properties.

The above list has increased since the publication of the first survey on FL [35], which focused on improving the communication efficiency of a very inefficient embryonal version of FL. A more recent important survey [43] lays its foundation of the peculiar characteristics of FL, defining which unique challenges it poses to the research community. The origin of these is an intriguing question: how to train a model on data distributed across multiple clients without ever collecting it in a single location or seeing it? The research community agreed that this conundrum causes FL to be so challenging. In addition, the number of elements composing a natural FL system is massive, making this research topic inherently multidisciplinary. Nonetheless, the evident difficulty of the problem did not stop the research community from proposing many solutions to it, mainly because FL potentially allows ML models to reach an unprecedentedly massive amount of data. Scaling up FL to worldwide populations of devices has been of particular interest to the research community [6].

2.1.1 Averaging-based aggregation algorithms

This section will focus on the most critical stage of FL training, i.e., aggregation. The aggregation stage is the most critical one since it is the one that allows the global model to improve its knowledge about the problem from the updates of the clients. The most popular aggregation algorithm is Federated Averaging (`FedAvg`) [53], the first algorithm proposed for FL. `FedAvg` is a synchronous algorithm that averages the updates of the clients giving them a weight that depends on the amount of data samples they trained locally. `FedAvg` achieves some degree of communication efficiency when increasing the amount of training performed locally by the clients and reducing the number of communication steps necessary to reach convergence. However, it is not robust to the heterogeneity of the clients [2], often referred to as non-IIDness, which is a common characteristic of FL systems [22]. Non-IID clients' updates can drift too much from the global model yielding a slower convergence or even divergence in some extreme cases, as shown in [93].

The research community has proposed diverse strategies to tackle this problem with frameworks that promote fairness and robustness, the most relevant of which are mentioned in the following. Every successful aggregation algorithm builds upon `FedAvg` and relies on the averaging procedure. `FedProx` [44] is a synchronous algorithm that introduces a proximal term to the local objective function of the clients, which penalises the distance of the local model from the global one. This penalisation term prevents clients from drifting too far from the global model while training locally, guaranteeing a more stable aggregation process at the server. This strategy implicitly assumes the heterogeneity of clients to be a weakness of the system and not a source of more information to learn. In [45], the authors propose to change the optimisation objective to tackle fair resource allocation in FL, inspired by the literature on wireless networks. `Q-FedAvg`, a communication-efficient variant of `FedAvg`, can solve their novel objective by encouraging a more uniform distribution of accuracy across clients, the definition of fairness proposed in the paper. The fairness objective proposed overcomes previous proposals in the literature of wireless networks and federated learning [91, 55]. The underlying idea of this strategy is to give more weight to the updates of the clients with a higher test loss. Although `q-FedAvg` is very effective, it requires an additional hyperparameter q to govern the weighing procedure, which is tricky to tune.

Another interesting path of research has explored the possibility of using control variates to reduce the variance of the updates of the clients. In [32], the authors propose `SCAFFOLD` that requires the clients and the server to keep a state of the training, the control variates, that is used during the aggregation procedure to limit the *client drift* per client. Acting at a client level allows one to discriminate between different clients, achieving robustness. `SCAFFOLD` has many advantages regarding final accuracy compared to previous baselines, but it needs to exchange more information. In addition, it requires the clients to keep a state of the training, which is not always possible in FL systems. Inspired by centralized adaptive optimisation methods, in [72] the authors propose the federated version of state-of-the-art adaptive optimizers such as `AdaGrad` [13], `Adam` [34], and `Yogi` [92]. The first attempt to leverage a momentum mechanism in optimising FL aggregation is in [23], where the authors proposed `FedAvgM`. While in `FedAvgM`, the only extension to `FedAvg` is in a learning rate at the server side for the global update computed after the aggregation, the authors of [72] proposed a more extensive formulation. They, indeed, define the federated optimisation, `FedOpt`, by splitting the two main steps of the optimisation procedure, i.e., local training and aggregation, which are called `ClientOpt` and `ServerOpt`, respectively. This procedure naturally accounts for two different learning rates, η_l and η_s, used to update the local and global models, and it also enables adaptive optimisers on both the client and the server sides. `FedOpt` comes for making accessible the tuning of the hyperparameters of the optimisers, which are difficult to tune in a federated setting. However, it still relies upon the averaging procedure with widely known performance issues when used under non-IID settings.

In [46], the author took another path for providing fairness and robustness in FL optimisation. They cast the optimisation problem to the empirical risk minimisation (`ERM`) that tackles estimating a parameter of interest by minimising the average loss over the data. Since `ERM` relies on averaging the loss, it can be highly susceptible to outliers lacking generalisation, resulting in unfairness. The authors propose tilted empirical risk minimisation (`TERM`) that extends `ERM` with the addition of a hyperparameter, called *tilt*, and that can flexibly address the problems of `ERM` while retaining its statistical properties. In [47], the authors address the problem of client heterogeneity by leveraging the personalisation of the models for the clients. They propose `Ditto`, a bi-level optimisation problem for each client in the federation with a multi-task framing. Despite they showed `Ditto` to be effective in achieving fairness and robustness, it has a couple of drawbacks, such as it relies upon the averaging procedure and each client needs to store a version of its local model even when it does not participate the current round.

Another exciting path the research community has taken toward improving FL under non-IID settings is Sharpness Aware Minimisation (`SAM`) [18]. First in [66] with `FedSAM`, then in [76] with `FedSMOO`, different research groups tried to leverage `SAM` optimiser in local minimisation and both local and global minimisation, respectively. The main objective of these works was to devise an algorithm that was able to understand the curvature of the loss landscape for seeking flatter minima, which are known to provide for better generalisation [33, 16, 30]. While `FedSAM` is seeking an optimal flatter region in the loss landscape of the global model, `FedSMOO` makes a step toward personalisation, taking into account the locality of clients' optimisation by seeking a region in the loss landscape of the local models. Although these two methods outperform `FedAvg`, they cannot overcome the averaging procedure.

2.1.2 Large-Scale FL

The characteristics of FL allow academia and industry to possibly train an ML model on a single system of an unprecedentedly massive and distributed federation of edge devices, such as smartphones, tablets, laptops, wearables, and many more [49]. The edge computation on such a worldwide distribution can relieve much of the data centres' work and allow ML pipelines to run on an embarrassingly parallel setting with the drawback of increasing the carbon footprint [65]. While the research is spending some resources addressing the carbon footprint of FL [88], large-scale FL still lacks a deep analysis and characterisation. Even though the literature has implicitly treated large-scale FL in most works labelled as "suited for cross-device" settings, which peculiar components characterise this setting is still unclear. Most cross-device FL research works account for an evaluation set that can arguably mimic a large-scale scenario. In addition, many do not wholly assess the scalability of their proposed solution. Then, the research community has to develop a set of clear and well-defined baselines that characterise the large-scale FL setting.

In [6], the author suggest many guidelines for deploying large-scale FL system accounting for the challenges, the problems, and possible solutions. The federation considered in their work comprises tens of millions of real-world devices. At the same time, they account for a future in which the federation's size can break the roof of billions of devices. In this future scenario, the orchestrator will likely sample as many clients per round as possible to obtain the most extensive coverage potential of the entire population. The number of clients per round, often referenced as *cohort size*, is one of the most critical hyperparameters in FL training. As shown first in [8], even though the population is substantial, `FedAvg` does not benefit from increasing the *cohort size*. This drawback limits the capacity of FL to parallelise the work, consequently speeding up the convergence to the optimum. The consequences of this overhead could be disastrous since wasted time corresponds to wasted computing resources, energy, and money while increasing the application's carbon footprint. Finally, it is still unclear how to leverage large cohorts in FL to exploit its parallelisation capabilities wholly.

2.1.3 Next-gen FL applications

As federated learning matures, the research community starts addressing more complex scenarios. We can foresee future FL applications that rely on the recent advancements from centralised and decentralised ML research, such as multimodality, multitask learning, continual learning, asynchronous federated optimisations, exotic aggregation methods, and many more. In the following, we present insights about some of these emerging scenarios, focusing on the opportunities that large-scale FL offers for their development.

2.1.3.1 Multimodality

Multimodal models have gained popularity recently since the developments of large language models (LLMs) and the success of OpenAI's products, such as the GPT family [67, 68, 7, 64] and the multimodal models [69, 70, 61]. Multimodal models are a class of models that can process different data types, such as text, images, audio, and video. They have proven their capabilities by achieving state-of-the-art performance in different tasks, such as image classification, image captioning, text-to-speech, and many more. Although, training multimodal models is computationally expensive and requires a large amount of pre-processed data in the shape of tuple image-text. These multimodal datasets are challenging to collect and maintain, especially when needing labels.

Large-scale FL represents one of the most promising solutions to deal with both the challenges of computation expenses and the amount of data. On the one hand, it allows distributing the computation across millions of devices, allowing for a high degree of parallelisation. On the other hand, it allows private training of multimodal models on a massive amount of real-world data moving the cost of collecting and maintaining such data to the edge device. This research path cannot be navigated without addressing its many challenges. Indeed, it is still unclear how to deal with extensive models when training on constrained devices such as smartphones. In addition, most real-world clients possess incomplete data, which can lead to a significant performance drop.

The research community has been actively developing new compression techniques, as described in Section 2.3, to address the former challenge by reducing the model's size without affecting its performance. Others have proposed techniques based on split learning (SL) [75, 79, 78, 80] whose objective is to train a large model on a server while clients are collaborating in training a small part of it. The last challenge is more complicated than the first because we cannot see or touch local data by design in FL. Exciting techniques such as pseudo-labelling and self-supervised learning [19, 73, 21] address the problem of missing labels and data in FL. Another promising direction is that of zero-shot stitching between models from different architectures proposed in [57, 63] for the centralised setting. The opportunities for exploiting the latter research result in FL are evident since the method needs few data samples and requires no training.

2.1.3.2 Multitask learning

Multitask learning (MTL) is a machine learning paradigm that aims to improve the performance of a model by learning multiple tasks simultaneously. The idea is that since the model is forced to learn different tasks continuously, it will learn a more general representation of the data enabling fast adaptation to single tasks. The relevancy of MTL to FL is generated from the parallelism between clients and tasks. Indeed, we can interpret each client as a task, and each task can be trained on a different client, or, more loosely, clients can be grouped in a set of possible overlapping tasks.

In the context of large-scale FL, research results from the realm of MTL can be leveraged to improve the performance of FL models. A critical example is the first work proposing *gradient surgery* for MTL [89]. The authors identified the so-called "Tragic Triad": conflicting gradients, dominating gradients, and high curvature. Once defined, these conditions characterise the appearance of "gradients' interference" in MTL, which produces convergence problems. Rigorous mathematical definitions are building a framework for analysing this phenomenon and proposing solutions to it. The authors propose a model-agnostic gradient surgery technique, `PCGrad`, that uses projection and vector subtraction to correct interference between gradients. Despite the promising results, `PCGrad` implicitly selects a preferred gradient direction, which can lead to biased solutions. In addition, since the involvement of the subtraction operations, some information contained in the gradients is lost during the process. Although this work collocates in the MTL realm, it is very relevant to FL since the evident similarities between the two paradigms, especially from a phenomenological perspective.

As in MTL, FL involves the aggregation of gradients that may differ and interfere with each other. In [82], the authors tried to apply this framework to FL, proposing a trivial extension, called `FedFV`, to the original algorithm. Their proposal requires the addition of two hyperparameters to the standard FL setting, which represent two very difficult thresholds to tune. `FedFV` is competitive with alternative techniques tackling the same setting, such as `AFL` [55], `qFedAvg` [45], `FedFa` [26], `FedMGDA+` [24], but the evaluation is limited to a single dataset and a single model with a cross-silo setting. In [20], the authors try to extend and improve `FedFV` by proposing a hierarchical fair federated learning framework called `FedHF`. Using the same framework based on conflicting information, they do not propose any new algorithm but instead try to improve the performance of `FedFV` by introducing a hierarchical structure. We argue that the shared limitation of such methods is two-fold: first, they necessitate a bias in the gradient direction selection, and second, they eliminate the supposedly conflicting information wasting the opportunity to learn from it.

2.1.3.3 Asynchronous FL

The standard aggregation strategy in FL assumes asynchronous aggregation of clients' updates forcing the server to wait for all selected clients to complete their training step. When the federation is composed of heterogeneous devices having diverse computational and network capabilities, this strategy can lead to significant delays in the training process due to the so-called *stragglers*. *Stragglers* are devices that take longer than the selected peers in a federated round to complete their training step because either they are slower in training the NN model or need a long time to communicate with the server. It is important to note that the more the devices in the federation are heterogeneous, the more the problem of *stragglers* becomes relevant, so we expect that large-scale FL suffers from this problem.

Asynchronous FL (AFL) is a research direction that aims to address the problem of *stragglers* by allowing clients to communicate with the server at any time [87]. In AFL, clients can send the updates to the server as soon as they complete their training step and begin another one without waiting for the other clients to complete their training. This strategy allows the server to aggregate the clients' updates as soon as they are available, reducing waiting time and overall training time. The trivial way to aggregate updates in AFL is to sequentially aggregate updates to the local model as soon as they are available, but it results in slow and unstable convergence [87]. Every AFL procedure has to deal with the problem of *staleness*, which appears when a client is training an outdated global model compared to the current one.

The following will summarise the most relevant and popular AFL aggregation strategies. `FedBuff` [60] represents the first exciting work to propose an intelligent aggregation procedure. It uses a buffer to store the clients' updates and aggregates them batched while allowing a limited number of clients to train with possibly different versions of the global model. `FedBuff` requires the addition of two hyperparameters which are difficult to tune, and it does not account for the problem of *staleness* in a large-scale setting. `PAPAYA` [27] is the first production system to support both FL and AFL and to evaluate AFL at scale demonstrating the AFL represents the future of real-world large-scale FL. They obtained that AFL is faster, more communication-efficient and achieves more fair models than synchronous FL. `PAPAYA` is based on `FedBuff` inheriting its limitations and some of its open questions.

`AsyncDrop` [15] aims to implement the widely used `Dropout` technique in FL at a client level based on `HogWild!` method [71, 50] and Independent Subnetwork Training (IST) [14, 48, 83, 90]. They assign different subnetworks to different clients achieving time to convergence to a fixed accuracy level while accounting for devices' heterogeneity at clients. Although this method obtains good results in the particular context they evaluated, it has some limitations: it possesses the same hyperparameter as `FedBuff`; it relies on an obscure grouping scheme for the weights of the global NN; it has been evaluated on simulation setting that cannot be considered general nor large-scale.

The literature in AFL demonstrated to be a promising technique in large-scale FL, while the proposed methods only partially address the challenges of this particular setting.

2.2 GNNs in FL

Graph neural networks (GNNs) have emerged as potent tools for effectively handling graph-structured data [85]. In such data, samples are interconnected through a graph topology, where nodes represent entities (e.g., atoms in molecular data) and edges signify relationships (e.g., bonds between atoms). By leveraging the inherent graph structure, GNNs can enhance the precision of node embedding by incorporating pertinent neighbourhood information extracted from the underlying graph topology. This section will show that the research community has unexpectedly overlooked GNNs-driven aggregation despite enormous opportunities for exploiting them in FL. Indeed, having an underlying graph structure in FL, especially in the case of IoT devices, is quite common because of the geographical and social relationships emerging between clients. Even though this is not the case for every FL application, we can always construct a graph structure by exploiting the similarity between clients' pieces of information received during the federated training. This opportunity is supported by the fact that the most general information that the server receives from clients represents the locally trained model that we can embed in a topological space in which some predefine distance metric can give us a graph structure. Recently inn [51], the authors propose the first survey on the application of GNN to FL (`FedGNNs`). They define a 2-dimensional taxonomy depending on two orthogonal axes defining the intersection between FL and GNN. The first dimension discriminates between works in which GNNs assist FL and those in which FL facilitates GNNs training. Secondly, the authors distinguish between different works using GNNs alongside FL from the perspective of how they tackle the problem of heterogeneity across clients in FL. These two orthogonal viewpoints constitute the principles of the authors' 2-dimensional taxonomy.

We will constrain our discussion in this section to those works that use GNNs to drive the aggregation process in FL, even though the number of exciting works is very limited, demonstrating once again that this opportunity has been widely overlooked.

[84] [86] [9] [40] [37] [54]

Survey ion temporal graphs [52] Algebraic and Geometric Models for Space Networking [3]

2.3 Compression techniques and FL

One of the hot questions related to FL has always been how to deal with the communication overheads deriving from sending back-and-forth (large) NN models from server to clients. Compression techniques, e.g. pruning, tackle this problem by reducing the model size without affecting performance. The research community has also started looking at the relationship between these and FL for reducing models' sizes to match the computational capabilities of edge devices. In a large-scale setting, leveraging any means of reducing the size of models can simultaneously achieve the two abovementioned goals, but it is not always the case. As the following paragraphs will show, the recently proposed compression techniques applied to FL often assume a set of properties of the federation that prevent their application from being suitable to achieve both objectives in a large-scale setting. This section will present the most relevant compression techniques and their application to FL, highlighting their strengths and weaknesses.

2.3.1 Pruning: Lottery Ticket Hypothesis

Sparse Random Networks for Communication-Efficient Federated Learning [28] The authors propose Federated Probabilistic Mask training (FedPM), a strategy to train binary masks during FL while keeping the weights fixed. Masks are randomly initialised using Bernoulli distribution, one for each client. They use a Bayesian aggregation procedure which couples naturally with the Bernoulli distribution that initialises masks. The global network is initialised once with a pre-determined seed and a pre-determined distribution. This info will be broadcasted to clients without sharing the initial model parameters. FedPM learns the probabilities for the weights being active. They assume this method to be highly communication-efficient since they just broadcast a global seed for initialising the model. During the federated training, only the binary masks are exchanged. The process is more communication-efficient since the clients send a stochastic binary sample from their probability masks to the server. This sampling follows a Bernoulli distribution of the probability mask. The aggregation process satisfies determined error bounds because of the statistical properties of the Bernoulli distribution and the unbiased estimation procedure performed to get the average mask. In the Bayesian interpretation, the aggregation strategy corresponds to a posterior update. The implementation takes advantage of the conjugate relation between the Beta-Bernoulli distributions. This seems to be helpful only with the partial participation of clients. They use a heuristic schedule to reset the priors: 3 rounds. The initialisation of the network is crucial. They use uniform distribution with sigma coming from the st. dev. of the Kaiming Normal distribution to control the variance of the output activations to be close to one. Weights need to be saved into memory during inference, but since the framework they're using, for each weight, one would only need to indicate whether the value is $-\sigma, 0, +\sigma$. They claim this ternary representation can be efficiently deployed on the hardware. The evaluation section is excellent. It just may lack baselines. They use an unusual procedure for generating non-IID splits of the centralised datasets. There's some code for this.

LotteryFL: Personalized and Communication-Efficient Federated Learning with Lottery Ticket Hypothesis on Non-IID Datasets [38] The proposed approach is a personalised, communication-efficient, federated learning framework that exploits the Lottery Ticket Hypothesis. Each client learns a lottery ticket network (personalised model) communicated between clients and the server to achieve personalisation and high communication efficiency under non-IID settings. The local pruning is thresholded by a (global) target pruning rate and a (global) target accuracy. This results in local, personalised masks. The weights are rewinded after pruning, resulting in a Lottery Ticket Network (LTN). Only LTNs are sent to the server and aggregated. They use a quantitative metric called Client-Wise Non-IID Index (CNI) to measure the degree of non-IID-ness. The resulting overlap depending on the pruning rate is interesting, and they plotted the overlapping for each relevant layer of the neural network. They found that the pruning rate surprisingly doesn't affect accuracy. However, there are several limitations to the proposed approach, including the fact that a global model is not learned, just two datasets are proposed for evaluation (MNIST and CIFAR-10), and the entire process is repeated at every round. Additionally, choosing hyperparameters may become problematic in large-scale settings, heterogeneous or data-driven pruning rates haven't been explored, and the total number of clients is small. Finally, no confidence interval is shown in the results. It doesn't seem to be suitable for cross-device. LTN is learned by pruning using local data

FedLTN: Federated Learning for Sparse and Personalized Lottery Ticket Networks [59] The text describes the proposal of FedLTN, a method for learning sparse and personalised models for communication-efficient Federated Learning in non-IID settings. They build upon LotteryFL but state that it does not achieve the pruning performance of LTH. They address some of the issues of LotteryFL, such as the limitations derived from the accuracy threshold and their "unfair" evaluation choices. Pruning without rewinding is motivated by the averaging procedure, not suffering too much from overfitting. They propose Jump-Start, which involves training one client and using the obtained Lottery Ticket Network (LTN) for transfer learning at others. They argue that preserving batch normalisation statistics provides for more personalised models. Evaluation is limited to CIFAR-10 and Tiny ImageNet, with only 100 and 10 clients. The evaluation is ridiculous. It doesn't seem to be suitable for cross-device.

Efficient Federated Learning with Enhanced Privacy via Lottery Ticket Pruning in Edge Computing [74] The authors propose Fed-LTP, a privacy-enhanced FL framework with LTH and zCDP. They generate a pruned global model on the server side and conduct sparse-to-sparse training with zCDP on the client side. They propose two pruning techniques and suggest double-pruning the global model on public data before federated training. They also propose moving the second pruning step to the federation. The evaluation and final model selection procedures are criticised.

Complement Sparsification: Low-Overhead Model Pruning for Federated Learning [29] This text describes the Complement Sparsification (CS) method for pruning in Federated Learning (FL). CS aims to achieve low bidirectional communication overhead, low computation overhead at clients, and good model accuracy. The method involves complementary and collaborative pruning at the server and the clients. Both global and local sparse models are computed respectively at the server and clients. The global model captures general knowledge from the federation, while the sparse local models capture local trends at clients. Clients' sparse models are pruned from the global model. Local sparse models and the sparse global model are complementary since they are aggregated into a dense model at each round which is then pruned iteratively. The authors of this text highlight a set of properties that sparsification/pruning techniques must satisfy in an FL context: reduce the size of the local updates from the clients to the server; reduce the size of the global model from the server to the clients; reduce pruning computation at the clients; achieve comparable model performance to the dense model. However, they note that this setting does not account for the partial participation of clients at each round. During the first round, vanilla FL is performed (with full participation). The server receives the updates and aggregates them as in vanilla-FL. The server prunes the global model and sends the complementary of it to the client, i.e. the pruned weights. The clients then train on a global model whose zero weights are the weights non-pruned by the server. The clients send back the locally trained sparse models to the server. The server aggregates the updates, which should modify the weights that survived the previous pruning step. The server prunes the model again and repeats. Clients use the converged global sparse model for inference.

Adaptive Gradient Sparsification for Efficient Federated Learning: An Online Learning Approach citepDBLP:conf/icdcs/HanWL20 The authors propose a framework for applying adaptive gradient sparsity in a non-IID setting. Their method uses an estimated sign of the derivative of the objective function to automatically determine the near-optimal communication and computation trade-off controlled by the degree of gradient sparsity. Clients always keep accumulated local gradients in their online learning algorithm during federated learning. Clients return the top-k gradient elements taken from the accumulation to the server at each round. The server receives top-k gradient elements for each client and selects the top-k elements overall by keeping at least one gradient element per client (this is meant to ensure fairness). The server then sends the identified top-k gradient elements to clients for the update. It's to be noted that all the gradients are continuously accumulated during local training.

FedMask [39] server randomly init the model and distribute to clients; then each device learn a binary mask via one-shot pruning method; in each round, select client to optim the mark and then only communicate the updated binary mask; server aggregate the updated binary mask but only aggregate the overlapping (appear in 2 or more clients) ones and keeping the non-overlapping one unchanged (to maintain personalisation) optim the mask while keeping the model parameter unchanged prune the last layer Clients train the local model (in its entire size) while learning a binary mask based on pruning and structure sparsity regularisation exploiting their local data. Once the mask has been learned, they use it for further training and communication: the overheads for learning the masks are mitigated by faster training afterwards and very cheap communication since only the mask is communicated. Amazing results. Recent papers that cited this are very very interesting as well as relevant. Limitations: They don't cluster masks together at the server. They didn't explore any other compression technique. The mask remains constant after learning, and there are no guarantees that the mask can always be learned. Large-scale experiments are missing Extensions/Ideas coming out Studying how the performance of the systems varies while changing the threshold for the masks would be very interesting: Estimating how the overlap of information changes as the threshold change Treating binary masks as target probabilities, Variational Autoencoders (VAEs) Large-scale experiments are missing.

DisPFL: Towards Communication-Efficient Personalized Federated Learning via Decentralized Sparse Training citepDBLP:conf/icml/Dai0H0T22 peer-to-peer solution for saving communication They use personalised sparse masks to customise sparse local models on edge.

GossipFL: A Decentralized Federated Learning Framework With Sparsified and Adaptive Communication [77] similar to the above one

EB-FedAvg: Personalized and Training Efficient Federated Learning with Early-Bird Tickets [41] Based on Early-Bird Tickets, each client learns a subnetwork of the base model, which is then communicated between the server and clients, reducing the communication costs. Each client thus learns a personalised model.

SPinS-FL: Communication-Efficient Federated Subnetwork Learning [81] They first consider a naive FL based on the Edge-Popup algorithm [10]. The Edge-Popup-based neural network constructs a subnetwork by assigning a score to each randomly initialised weight and selecting weights based on the score without updating the weights. Based on the subnetwork consisting of only the live weights with high scores, forward propagation is performed. Then backpropagation is performed to correct scores and search for a subnetwork that achieves high accuracy. The proposed SPinS-FL only communicates some supermasks and a partial set of scores of the weights around the boundary between selected and unselected in the subnetwork; this dramatically improves communication efficiency.

TailorFL: Dual-Personalized Federated Learning under System and Data Heterogeneity [11] TailorFL is a dual-personalized FL framework which tailors a submodel for each device with a personalised training structure and parameters for local inference. They propose a resource-aware and data-directed pruning strategy that makes each device's submodel structure match its resource capability and correlate with its local data distribution. They design a scaling-based aggregation strategy that scales parameters with the pruning rate of submodels and aggregates the overlapped parameters. Ultimately, they propose a server-assisted model-tuning mechanism, which dynamically tunes the device's submodel structure at the server side with the global view of the device's data distribution similarities.

Towards Mitigating Device Heterogeneity in Federated Learning via Adaptive Model Quantization [1] They propose AQFL, a simple and practical approach leveraging adaptive model quantisation to homogenise the computing resources of the clients.

Federated Learning with Online Adaptive Heterogeneous Local Models [94] They establish sufficient conditions for any FL algorithms with heterogeneous local models to converge to a neighbourhood of a stationary point at a determined rate. Their analysis illuminates two key factors impacting the optimality gap between heterogeneous and standard FL: pruning-induced noise and minimum coverage index, advocating a joint design strategy of local models' pruning masks in heterogeneous FL algorithms.

Fusion of Global and Local Knowledge for Personalized Federated Learning [25] They explore personalised models with low-rank and sparse decomposition. Specifically, we employ proper regularisation to extract a low-rank global knowledge representation (GKR) to distil global knowledge into a compact representation. Then they employ a sparse component over the obtained GKR to fuse the personalised pattern into the global knowledge. They propose a two-stage proximal-based algorithm named Federated learning with mixed Sparse and Low-Rank representation (FedSLR) to search for the mixed models efficiently. They show that the GKR trained by FedSLR can at least sub-linearly converge to a stationary point of the regularised problem. The sparse fused component can connect to its fixed point under proper settings.

Model Elasticity for Hardware Heterogeneity in Federated Learning Systems [17] They argue that since most FL works rely on using the same architecture for all clients, the impact of hardware heterogeneity has been overlooked. They propose a model-architecture co-design framework based on the concept of model elasticity. Clients select different models of the same architecture family depending on their resource budget. This is performed by leveraging elastic transformations that are invertible functions mapping clients' models to the big global model. One elastic transformation per client is built and stored in a lookup table. Sparse local models are created using IMP. The authors propose elastic Learning for Federated systems (eLF). The empirical analysis of the proposal is in a cross-silo setting.

FRL: Federated Rank Learning [58] The authors aim to tackle the problem of poisoning attacks in FL, stating that one primary source is in using large models. They propose Federated Rank Learning (FRL) that reduces the space of client updates from the model parameter updates to the space of parameter rankings. They don't leverage IMP/LTH. Instead, they use the

Edge-popup (EP) algorithm [10] which consists in popping away edges in the large network to find supermasks. Clients don't communicate masks or updates, but they communicate rankings of the edges. The rankings are aggregated through a voting procedure in which each client weighs as a single voter. The interesting bits about this paper are the EP algorithm applied in FL and the voting procedure that seems to be robust. They achieve a very good communication performance even if their evaluation setting is closer to cross-silo than to cross-device. Also, the non-IID setting tested is a bit weak. NO CI IN THE RESULTS! There's room for extending the voting procedure to personalisation, although the author didn't investigate this aspect. I think we could also change from EP to IMP.

2.3.2 Quantization

2.3.3 Split learning and other techniques

2.4 FL frameworks

The development of FL frameworks is a crucial step toward democratising FL. The research community has developed many frameworks for FL, many of which are open-source. Future research addressing large-scale scenarios must rely on scalable, flexible frameworks that guarantee fast prototyping and quick deployment. This section presents the most relevant FL frameworks, highlighting their strengths and weaknesses. The focus is on simulation frameworks that are open-source and that are still under active development. The simulation FL frameworks aim to enable experiments in a constrained environment where virtualising all the clients or all the clients in the sampled cohort for each round may not be feasible [12, 36, 4]. These engines represent each client's training as a scheduled job, which includes allocating resources for model training, client-specific datasets, pre-processing, and training instructions. Since the resource needs of a single client are usually small, it is possible to fit multiple clients into a single GPU [4].

To orchestrate the training process, FL simulation engines adopt a server-workers paradigm. The server serves clients to workers, and after each round of training, it aggregates the results [12, 36, 4]. The workers, statically allocated to their GPUs in FedScale and Flute [36, 12], are responsible for training individual clients and sending the trained models to the server upon completion. On the other hand, Flower's Virtual Client Engine allows dynamic worker allocation using Ray [4, 56], but with limited control over the dynamic allocation.

In this framework, the server employs a pull-based queuing system for serving clients to workers, which involves multiple communication steps between the server and workers [12, 36, 4]. The execution of an FL round in this system can be summarised as follows:

1. At the start of a round, the server samples a subset of participating clients, defining the cohort for that round. The cohort is stored in a synchronised queue, allowing workers to read clients sequentially.

2. Each worker reads from the queue, extracts the first client and initiates the training process. Different workers cannot access the same client concurrently.

3. Once a worker completes training a client, it notifies the server.

4. Upon receiving the notification, the server informs the worker when it can receive the training results.

5. The worker then sends the results to the server, which stores or partially aggregates them, depending on the experiment's configuration.

The number of workers controls the concurrency of this embarrassingly parallel simulation. Hence, increasing the number of workers is advantageous until the system's resources are saturated or the gains from concurrency diminish [4].

However, pull-based queuing systems in federated learning frameworks have several limitations. First, the inability to choose which client to train can hinder the simulator's ability to allocate specific clients to particular GPUs [12]. Balancing client training time across GPUs and avoiding stragglers becomes challenging when disproportionately large clients are selected. Another general limitation is the significant communication overhead relative to the training time of most clients, especially in multi-node setups where network connections are required [12]. Flute, optimised for GPU training using the *nccl* backend of PyTorch Distributed [42], can run only a single worker per GPU [12]. It requires a dedicated GPU for the parameter server, responsible for aggregation during FL simulation, which can be wasteful as aggregation is typically not computationally intensive. Addressing these issues in Flute is challenging due to the highly coupled components of the codebase. FedScale relies on gRPC with unreliable configurations, resulting in potential crashes during longer rounds due to client disconnections or timeouts [36]. Although it allows multiple workers on the same GPU, increasing the number of workers or GPUs does not provide substantial benefits [36]. Additionally, each worker is tasked with loading the entire dataset, even if they share the memory with other workers on the same node. Like Flute, the codebase of FedScale has a high coupling level, making refactoring difficult. Flower, built upon Ray [56], may encounter out-of-memory (OOM) issues in a multi-GPU configuration because workers do not deallocate memory from previously used GPUs [4]. Careful configuration of parameters can mitigate OOM at the cost of slower overall training. However, Flower's highly modular codebase allows for implementing a new simulation engine on top of existing components.

Chapter 3

Proposed Research

The background chapter describes the current state of the art in the field of FL. It focuses on current solutions' impact in large-scale scenarios requiring a particular set of assumptions and properties. The most exciting research question we try to address in this proposal is *how can we take advantage of the entire amount of information that large-scale FL offers in a client-efficient manner.* As depicted in the background chapter, large-scale FL allows us to train ML models on an unprecedentedly large amount of data through a massively populated federation of clients. Thus, this setting opens novel opportunities for the modelisation of real-world data distributions and for developing more robust and generalisable federated ML models. However, the literature has not yet discovered a *client-efficient* strategy to leverage this information since the current state of the art is limited to averaging-based aggregation algorithms that present the phenomenon of updates cancelling out with each other. The answer to this question is not straightforward and requires a deep understanding of the field's current state of the art. We argue that this exciting research question splits into the following sub-questions:

- **Can we build an efficient, scalable, and fast FL simulations framework?** Researchers must test their ideas in a realistic setting to understand their impact and limitations. We want to investigate a challenging setting because of its properties, such as the size of the federation, the stateless nature of clients, the heterogeneity of the clients' computational power, the heterogeneity of data distributions across clients, and the privacy-preserving feature that federated learning has by design. This question is detailed in Section 3.1.
- **Can we leverage compression techniques to enable better models in large-scale FL?** Between the properties that clients in large-scale FL have, the heterogeneity of the hardware running the clients' training is one of the most challenging to deal with. Compression techniques can help to mitigate this problem by reducing the size of the models achieving a two-fold objective: matching the edge

device computational capabilities and reducing the amount of data that clients need to exchange with the server. In addition, we argue that compression techniques can help better to model the data distributions' heterogeneity across clients resulting in a better performance of FL models. Section 3.2 presents the details of this research question.

- **Is it possible to devise novel aggregation algorithms exploiting GNNs and deep learning's theoretical results?** The literature suggests that averaging-based aggregation algorithms are not the best option for efficiently collecting the entire amount of information the server receives from clients in large-scale FL. In addition, many novel ML techniques have reached their maturity in the last few years, primarily when associated with big-data contexts. The research community has overlooked the integration of these techniques, in particular GNNs, as a way of driving the aggregation process in FL. GNNs represent a family of techniques aiming to train ML models on a graph-based topology leveraging the connectivity between nodes to define the best function for aggregating the information exchanged through the graph. Section 3.3 depicts how we will address this research question.

- **Can we design a general framework for dealing with large-scale FL?** The previous research questions are all related to the same problem: how can we take advantage of the entire amount of information that large-scale FL offers client-efficiently? Ultimately, the audacious research question we tentatively address is to design a general framework for large-scale FL, which accounts for the contribution and achievement represented by the answers to the previous questions. We will certainly consider the developments in the field of FL and other exciting results that may arise during the research. We will also track the main research direction in the centralised ML field to investigate possible integration with FL. Section 3.4 describes the details of this research question.

The solution space we want to explore spans from compression techniques to alternative aggregation algorithms exploiting more exotic approaches such as clustering techniques, graph neural networks, and multi-task learning. The research areas mentioned above represent promising paths of investigation to achieve the audacious objective of proposing a general framework for large-scale FL. During this research, we will inevitably face many challenges and possibly clarify more minor research questions such as: what is the relation between FL and MTL, how to leverage compression techniques in large-scale FL, how can we use large-scale models in FL, how addressing the problem of conflicting information can improve performance in large-scale FL.

This chapter describes the proposed research. It splits into four main parts that we present as follows: the design of an efficient FL simulation framework, the impact of compression techniques in such settings, the use of GNNs to drive the aggregation, and a general framework to deal with it.

3.1 Designing an FL simulation framework

In the context of FL research, relying on a scalable and efficient platform for developing simulations and eventually producing FL algorithms and pipelines is crucial. To match the requirements of the research community in the area, we designed a flexible and scalable FL framework that can run on both GPU clusters and real edge devices, such as Raspberry Pi. When dealing with a large-scale scenario that involves training up to tens of thousands of clients per round, an FL framework's design and capabilities can make a big difference. The challenging setting we want to explore imposes the framework to be resource-efficient, whatever the hardware setting is available to the researcher. It is often the case of having a multi-node multi-gpu cluster in which heterogeneous GPU devices plug into heterogeneous nodes with different SoCs and network capabilities. Another critical point to highlight is that the unitary component of simulated FL training is the local training of the client's data, which is a lightweight task compared to centralised counterparts. While in the centralised setting training a NN model requires many GPUs to collaborate in the computation, a task in simulating FL can run concurrently with its peers on a single GPU. The emerging problem becomes how to efficiently train a series of FL rounds composed of thousands of tasks (clients' training) on a constrained and heterogeneous hardware setting composed of a limited number of GPUs on a limited number of nodes. While many tasks can oversubscribe a single GPU, one powerful GPU, e.g. Nvidia A40, can host a few tens of such tasks concurrently, but an FL round needs thousands of tasks to complete. However, these minor tasks are heterogeneous since they usually take different world clock times to complete depending on the amount of data samples the clients have.

Our solution to this problem is a novel design for FL simulation engines that relies on a *push-based* placement of sequences of tasks to the GPUs available given the hardware setting. The placement strategy addresses executors that are isolated processes running concurrently on the GPU by sending them a list of clients to train sequentially. The composition of such lists is optimised to let the executors finish their list almost at the same world-clock time resulting in minimal waste of compute time and, eventually, an overall faster simulation. Our proposal accounts for heterogeneous GPUs and multi-node hardware settings, in which it achieves state-of-the-art performance compared to other FL simulation frameworks. With this novel design, we can also better understand the efficiency trade-off emerging when scaling up the number of clients per round or GPUs. This trade-off can be summarised in three results: the more constrained the resources are, the more efficient our system is; the more heterogeneous the resources are, the more efficient our system is; the larger the number of clients per round is, the more efficient our system is.

3.2 Compression enabling better modelling

As mentioned earlier, real-world FL applications require a deep consideration of the edge devices' capabilities in training ML models. Compression techniques are a promising solution to let ML training procedures match such capabilities and concurrently reduce the computation and communication overheads of FL systems. In this part of the research, we will explore these techniques that usually address devices' heterogeneity while adding one more feature: the ability to improve the modelling of the system given the data heterogeneity. We believe that data-driven compression techniques are also suitable for overcoming the issues of averaging procedures in large-scale FL, i.e., wasting precious information from clients' updates. As they consider the system's characteristics of the clients, they could additionally cast the information clients send to the server more meaningfully. Even though this context may seem very abstract, the practice suggests room for exploring this possibility. Indeed, previous works taught us that compression techniques let each client select the most relevant global model parameters to update related to the local data distribution when the method is data-driven. From this perspective, the challenges these methods have in leveraging averaging-based aggregation will conduct us to develop algorithms that may deviate from state-of-the-art. This research will thus address the question of how compression techniques such as iterative magnitude pruning (IMP), edge-popup (EP), powerpropagation, and others can benefit large-scale FL. This exploration will involve, but not be limited to, the utilisation of emerging techniques in FL, such as multi-modal models, asynchronous FL, and exotic aggregation procedures.

The proposal is to exploit a data-driven compression technique to provide clients with a suitable compressed version of the global model. It is essential to let the client have different compression rates and to treat them as ephemeral entities that may participate in the training procedure for a limited number of rounds with an unpredictable schedule. During this research, compression techniques will account for the additional objective of improving the global knowledge that the models have about the federation. In addition, we will consider both the global generalisation performance of the global model at the server and the personalisation performance at the clients. The evaluation of the proposed solution will be conducted on real-world datasets and ML models while considering intelligent modelling of the system's characteristics of clients and servers. We will possibly consider our solution's privacy implications by considering metrics such as the privacy budget, privacy leakage, and privacy loss.

3.3 Leveraging GNNs for aggregation

Previous works presented insights describing the limits that averaging-based aggregation algorithms, such as `FedAvg`, have when dealing with large cohorts of clients per round. The straightforward solution to these limitations has been to reduce the cohort size while running more federated rounds. Real-world systems rely on technologies that allow easy scaling up the number of clients per round to thousands of clients. Moreover, we foresee future FL applications having more than one level of aggregation, taking inspiration from many multinational service providers, e.g. Netflix, that distribute communicating controller nodes to serve different geographical regions. This future scenario could increase the number of clients per round that a federation could train on by orders of magnitude while dealing with a non-trivial hierarchical aggregation structure. These ingredients alone allow leveraging graph-based aggregation strategies in large-scale FL. Graph Neural Networks (GNNs) are very effective in learning aggregation functions on a graph structure, even when the graph topology is a trivial star-like structure. They even offer the chance to solve more complex situations in FL represented by ephemeral clients, partial participation, and hierarchical aggregation. GNN can leverage techniques to address these challenges: link prediction, higher-order aggregation, temporal GNN, and others. In addition, it is often the case that clients are inserted in an intrinsic topology, possibly defined by their geographical location or their social connections. During this research step, we plan to investigate how to leverage GNNs to drive the aggregation step in an FL round. This step splits into sub-questions that will comprise:

- *Which is the best representation clients have to use as a set of features in the GNN nodes?* The crucial challenge here is represented by the limited amount of information we can exchange in the context of FL. The literature considers exchanging the model trained on local devices safe, but this might represent information with too high dimensionality. We can use compression techniques, dimensionality reduction techniques, and geometrical tools to give more context and a better interpretation to locally trained models in a complex aggregation situation such as that of large-scale FL.

- *How can we represent the ephemeral clients or the partial participation of clients during a federated round?* GNNs support link prediction techniques and temporal graphs. We can leverage both these settings when dealing with partial participation, i.e. when only a subset of (randomly) selected clients participates in the current round of training, and ephemeral clients, i.e. when the entire population of clients may change during the training by deleting or adding clients on the fly.

- *Can we leverage GNNs for hierarchical aggregation in FL?* GNNs are offering interesting results in solving higher-order aggregation functions. Hierarchical federated learning represents a natural playground to explore the compatibility between GNNS and FL.

3.4 General framework for large-scale FL

Finally, the most ambitious part of this research is to devise a general framework for large-scale FL. This framework will be based on the results of the previous sections and will be updated with the latest developments in the field. Indeed, it is essential to consider the emerging properties of large-scale FL and to leverage novel developments in the field. It can be applied to real-world applications (datasets, models, and systems), possibly estimating privacy trade-offs. In addition, a particular focus will be on the communication-computation trade-off, which is crucially challenging due to the heterogeneous needs that edge devices have. A well-designed framework will enable multimodal and next-gen applications, such as interleaving and concurrent multi-task FL.

Indeed, the literature comprises plenty of solutions aiming to address large-scale scenarios that rely on a set of conditions that are not likely to appear in the real world, such as full participation of clients, complete identification of clients, stateful clients, homogeneous computational power, homogeneous network connections, and homogeneous local datasets. This part of the research aims to fill this gap by assuming the abovementioned properties and considering all the components involved from both a system and an algorithmic perspective. This part will consider the following sub-topics:

- **General system characterisation.** The first sub-topic aims to define a set of metrics to characterise large-scale FL systems. We will consider the assumptions based on the setting and pay particular attention to the diverse clients' needs in such settings. It is unreasonable to treat all of them as peers as they may have different system characteristics regarding computational power, memory, and network bandwidth. It is critical to underline patterns and define the particular characteristics of these settings compared to the standard categorisation used in the literature, which limits to discrimination of cross-device from cross-silo FL. Real-world datasets, collecting systems characteristics and federated data partitions, are open-source and available on the web, and more can be collected to provide a baseline for this research. Our group is deeply involved in collaborating with the industry to collect real-world datasets.

- **Impact of conflicting information.** Secondly, we aim to address the impact of conflicting information in large-scale FL, which depends on two main components: the nature of the piece of information clients are sending back to the server and the aggregation procedure used to build the global knowledge with such pieces. The literature has already investigated the impact of conflicting information in MTL and partially in FL but has not considered the large-scale scenario. However, it has always restricted the analysis at investigating gradient updates aggregated through `FedAvg`. Geometrical tools, theoretical approaches for deep learning, and previous findings will help us during this stage. The literature now uses tools like cosine similarity metric and dimensionality reduction techniques to deal with the updates from clients whose dimension is usually the same as the large global model.

- **Impact of large-scale models.** The literature is just starting to address the impact of large-scale models in FL. Even though these models are too big to be trained in a single device, many solutions for enabling their usage in FL have been explored, such as split learning, compression techniques, and adaptors. The relevance of this step relies on the fact that since large-scale models are often pre-trained on a large-scale, possibly curated, dataset, they may already have a good understanding of the data distribution of the federation. For example, the embedding layer of a pre-trained large language model (LLM) represents the perfect frame of reference for dealing with the data heterogeneity of clients without leaking sensitive information since it is built on a publicly available dataset.

Chapter 4

Timeline

Term	**Planned work**
Summer 2023	Improve the work of the FL simulation engine. Propose *Pollen+*, an enhanced version, empowered by a continuous performance profiler, of the previous work supporting Vertical FL, Asynchronous FL, stateful clients, and re-placement on-the-fly.
Michaelmas Term 2023	Explore compression techniques for large-scale FL. Exploring heterogeneous architecture with aggregation enabled by relative representation is a work in progress. Preliminaries on how to use GNNs in FL.
Lent Term 2024	Investigate the relationship between clients' heterogeneity and the local models obtained through compression techniques. Continue preliminaries on how to use GNNs in FL.
Easter Term 2024	Propose a compression-enabled framework for large-scale FL. Explore GNN-driven aggregation.
Summer 2024	Internship.
Michaelmas Term 2024	Propose a novel GNN-driven aggregation method for large-scale FL. Wrap up the knowledge obtained to delineate a tentative design for a general large-scale FL framework.
Easter Term 2025	Design and tentatively implementing a general large-scale FL framework.
Summer 2025	Thesis outline. Complete unfinished work. Thesis write-up.
Michaelmas Term 2025	Thesis write-up.

Table 4.1: Timeline of the PhD research proposal.

Chapter 5

Completed work

During the first year of PhD, we started investigating the problems related to large-scale federated learning. The main focus was on enabling future research by developing an efficient and scalable federated learning framework. The assumptions we made for designing the framework were focused on providing developers with a flexible tool that can easily switch from research to production with minimal effort. It must support the broadest range of federated learning algorithms and scale to millions of devices while supporting the broadest possible set of systems. In this section, we attach two research works in which we have made significant progress. The first paper [5] presents the Flower framework submitted and rejected at *MLSys 2021?*. Despite obtaining a rejection, Flower is now a popular open-source framework for federated learning used by thousands of developers worldwide for both research and production. A large community of contributors supports its continuous development while the leading developers working on enabling Flower with the most recent technologies in the field. The second paper presents Pollen, a work submitted and rejected at *Mobicom 2023*. Pollen empowers the simulation engine in the Flower framework with an efficient placement strategy that aims to exploit, at the best of its capabilities, the hardware available. It achieved an incredible performance in terms of world clock time compared to other frameworks while supporting heterogeneous hardware (multi-GPUs settings) and multi-node settings.

Flower: A Friendly Federated Learning Framework

Daniel J. Beutel [1 2] Taner Topal [1 2] Akhil Mathur [3] Xinchi Qiu [1] Javier Fernandez-Marques [4] Yan Gao [1]
Lorenzo Sani [5] Kwing Hei Li [1] Titouan Parcollet [6] Pedro Porto Buarque de Gusmão [1] Nicholas D. Lane [1]

Abstract

Federated Learning (FL) has emerged as a promising technique for 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 the data in the cloud. However, FL is difficult to implement realistically, both in terms of scale and systems heterogeneity. Although there are a number of research frameworks available to simulate FL algorithms, they do not support the study of scalable FL workloads on heterogeneous edge devices.

In this paper, we present Flower – a comprehensive FL framework that distinguishes itself from existing platforms by offering new facilities to execute large-scale FL experiments, and consider richly heterogeneous FL device scenarios. Our experiments show Flower can perform FL experiments up to *15M in client size* using only a pair of high-end GPUs. Researchers can then seamlessly migrate experiments to real devices to examine other parts of the design space. We believe Flower provides the community a critical new tool for FL study and development.

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

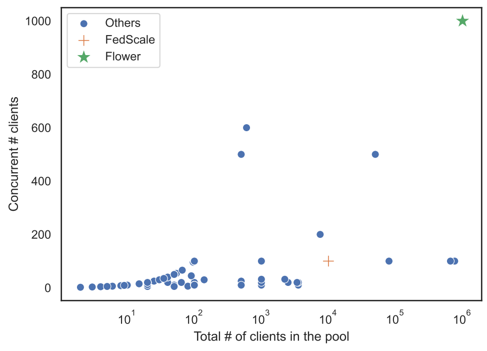

Figure 1. Survey of the number of FL clients used in FL research papers in the last two years. Scatter plot of number of concurrent clients participated in each communication round (y-axis) and total number of clients in the client pool (x-axis). The x-axis is converted to log scale to reflect the data points more clearly. FedScale can achieve 100 concurrent clients participated in each round out of 10000 total clients (orange point), while Flower framework can achieve 1000 concurrent clients out of a total 1 million clients (green point). The plot shows that Flower can achieve both higher concurrent participated client and larger client pool compared with other experiments existing the the recent research papers. Appendix A.1 gives details of the papers considered.

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 several frameworks including Tensorflow Federated (Google, 2020; Abadi et al., 2016a) (TFF) and LEAF (Caldas et al., 2018) enable experimentation on FL algorithms, they do not provide support for running FL on

[1]Department of Computer Science and Technology, University of Cambridge, UK [2]Adap, Hamburg, Hamburg, Germany [3]Nokia Bell Labs, Cambridge, UK [4]Department of Computer Science, University of Oxford, UK [5]Department of Physics and Astronomy, University of Bologna, Italy [6]Laboratoire Informatique d'Avignon, Avignon Université, France. Correspondence to: Daniel J. Beutel <daniel@adap.com>.

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 algorithms, 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 Figure 1. Even those papers which use more than 100 clients rely on 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 large-cohort 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:

- We present Flower, a novel FL framework that supports large-cohort training and evaluation, both on real edge devices and on single-node or multi-node compute clusters. This unlocks scalable algorithmic research of 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 algorithms, training strategies and communication protocols.
- Using Flower , we present experiments that explore both algorithmic and system-level aspects of FL on five machine learning workloads with up to 15 million 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 major research organizations in both academia and industry. The community is actively participating in the development and contributes novel baselines, functionality, and algorithms.

[1]https://flower.dev

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., 2019). Finally, there is an increasing emphasis on the performance of federated optimization 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., 2018) propose open source frameworks which are primarily used for simulations that run a small number of *homogeneous* clients. 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 *heterogeneous* devices.

3 Flower Overview

Flower is a novel end-to-end federated learning framework that enables a more seamless transition from experimental research in simulation to system research on a large cohort of real edge devices. Flower offers individual strength in both areas (viz. simulation and real world devices); and offers the ability for experimental implementations to migrate between the two extremes as needed during exploration and development. 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

The identified gap between FL research practice and industry reports from proprietary large-scale systems (Figure 1) is, at least in part, related a number of use cases that are not well-supported by the current FL ecosystem. The following sections show how Flower enables those use cases.

Scale experiments to large cohorts. Experiments need to scale to both a large client pool size and a large number of clients training concurrently to better understand how well methods generalize. A researcher needs to be able launch large-scale FL evaluations of their algorithms and design using reasonable levels of compute (e.g., single-machine/a multi-GPU rack), and have results at this scale have acceptable speed (wall-clock execution time).

Experiment on heterogeneous devices. Heterogeneous client environments are the norm for FL. Researchers need ways to both simulate heterogeneity and to execute FL on real edge devices to quantify the effects of system heterogeneity. Measurements about the performance of client performance should be able to be easily collected, and deploying heterogeneous experiments is painless.

Transition from simulation to real devices. New methods are often conceived in simulated environments. To understand their applicability to real-world scenarios, frameworks need to support seamless transition between simulation and on-device execution. Shifting from simulation to real devices, mixing simulated and real devices, and selecting certain elements to have varying levels of realism (e.g., compute or network) should be easy.

Multi-framework workloads. Diverse client environments naturally motivate the usage of different ML frameworks, so FL frameworks should be able to integrate updates coming from clients using varying ML frameworks in the same workload. Examples range from situations where clients use two different training frameworks (pytorch and tensorflow) to more complex situations where clients have their own device- and OS-specific training algorithm.

Table 1. Excerpt of built-in FL 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. (2020) to extend FL to heterogenous network conditions.
QFedAvg	Implementation of the algorithm proposed by Li et al. (2019) to encourage fairness in FL.
FedOptim	A family of server-side optimizations that include FedAdagrad, FedYogi, and FedAdam as described in Reddi et al. (2021).

3.2 Design Goals

The given uses cases identify a gap in the existing FL ecosystem that results in research that does not necessarily reflect real-world FL scenarios. To adress the ecosystem gap, we defined a set of independent design goals for Flower:

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.

Client-agnostic: Given the heterogeneous environment on mobile clients, Flower should be interoperable with different programming languages, operating systems, and hardware.

Communication-agnostic: Given the heterogeneous connectivity settings, Flower should allow different serialization and communication approaches.

Privacy-agnostic: Different FL settings (cross-devic, cross-silo) have different privacy requirements (secure aggregation, differential privacy). Flower should support common approaches whilst not be prescriptive about their usage.

Flexible: Given the rate of change in FL and the velocity of the general ML ecosystem, Flower should be flexible to enable both experimental research and adoption of recently proposed approaches with low engineering overhead.

A framework architecture with those properties will increase both realism and scale in FL research and provide a smooth transition from research in simulation to large-cohort research on real edge devices. The next section describes how the Flower framework architecture supports those goals.

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

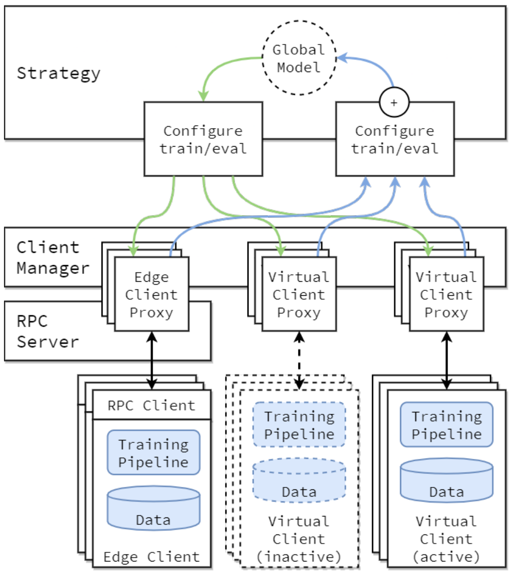

Figure 2. Flower core framework architecture with both Edge Client Engine and Virtual Client Engine. Edge clients live on real edge devices and communicate with the server over RPC. Virtual clients on the other hand consume close to zero resources when inactive and only load model and data into memory when the client is being selected for training or evaluation.

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. Global logic for client selection, 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) or FedYogi (Reddi et al., 2021) (summarized in table 1). 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 *Flower Protocol* level or using the high-level *Client* abstraction. Figure 2 illustrates those components.

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 *ClientManager*, the FL loop, and a (user customizable) *Strategy*. Server components sample clients from the *ClientManager*, which manages a set of *ClientProxy* objects, each representing a single client connected to the server. They are responsible for sending and receiving *Flower Protocol* messages to and from the actual client. The FL loop is at the heart of the FL process: it orchestrates the entire learning process. It does not, however, make decisions about *how* to proceed, as 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, receives the resulting client updates (or failures) from the clients, 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 simpler in the sense that it only waits for messages from the server. It then reacts 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 clients connected through different communication stacks. The framework manages underlying complexities such as connection handling, client life cycle, timeouts, and error handling in an for the researcher.

3.4 Virtual Client Engine

Built into Flower is the Virtual Client Engine (VCE): a tool that enables the virtualization of Flower Clients to maximise utilization of the available hardware. Given a pool of clients, their respective compute and memory budgets (e.g. number of CPUs, VRAM requirements) and, the FL-specific hyperparameters (e.g. number of clients per round), the VCE launches Flower Clients in a resource-aware manner. The VCE will schedule, instantiate and run the Flower Clients in a transparent way to the user and the Flower Server. This property greatly simplifies parallelization of jobs, ensuring the available hardware is not underutilised and, enables porting the same FL experiment to a wide varying of setups without reconfiguration: a desktop machine, a single GPU rack or multi-node GPU cluster. The VCE therefore becomes a key module inside the Flower framework enabling running large scale FL workloads with minimal overhead in a scalable manner.

3.5 Edge Client Engine

Flower is designed to be open source, extendable and, framework and device agnostic. Some devices suitable for lightweight FL workloads such as Raspberry Pi or NVIDIA Jetson require minimal or no special configuration. These Python-enabled embedded devices can readily be used as Flower Clients. On the other hand, commodity devices such as smartphones require a more strict, limited and sometimes proprietary software stack to run ML workloads. To circum-

Table 2. Comparison of different FL frameworks.

	TFF	Syft	FedScale	LEAF	Flower
Single-node simulation	√	√	√	√	√
Multi-node execution	*	√	(√)***		√
Scalability	*		**		√
Heterogeneous clients		(√)***	**		√
ML framework-agnostic		****	****		√
Communication-agnostic					√
Language-agnostic					√
Baselines			√	√	*

Labels: * Planned / ** Only simulated
*** Only Python-based / **** Only PyTorch and/or TF/Keras

vent this limitation, Flower provides a low-level integration by directly handling *Flower Protocol* messages on the client.

3.6 Secure Aggregation

In FL the server does not have direct access to a client's data. To further protect clients' local data, Flower provides implementation of both SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020) protocols for a semi-honest threat model. The Flower secure aggregation implementation satisfies five goals: usability, flexibility, compatibility, reliability and efficiency. The execution of secure aggregation protocols is independent of any special hardware and ML framework, robust against client dropouts, and has lower theoretical overhead for both communication and computation than other traditional multi-party computation secure aggregation protocol, which will be shown in 5.5.

3.7 FL Framework Comparison

We compare Flower to other FL toolkits, namely TFF (Google, 2020), Syft (Ryffel et al., 2018), FedScale (Lai et al., 2021) and LEAF (Caldas et al., 2018). Table 2 provides an overview, with a more detailed description of those properties following thereafter.

Single-node simulation enables simulation of FL systems on a single machine to investigate workload performance without the need for a multi-machine system. Supported by all frameworks.

Multi-node execution requires network communication between server and clients on different machines. Multi-machine execution is currently supported by Syft and Flower. FedScale supports multi-machine simulation (but not real deployment), TFF plans multi-machine deployments.

Scalability is important to derive experimental results that generalize to large cohorts. Single-machine simulation is limited because 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. FedScale can simulate clients on multiple machines, but only scales to 100 concurrent clients. Syft is able to communicate over the network, but only by connecting to data holding clients that act as servers themselves, which limits scalability. In Flower, data-holding clients connect to the server which allows workloads to scale to millions of clients, including scenarios that require full control over when connections are being opened and closed. Flower also includes a virtual client engine for large-scale multi-node simulations.

Heterogeneous clients refers to the ability to run workloads comprised of clients running on different platforms using different languages, all in the same workload. FL targeting 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-agnostic and *communication-agnostic* client-side integration points. It is the only framework in our comparison that does so, with TFF and Syft expecting a framework-provided client runtime, whereas FedScale and LEAF focus on Python-based simulations.

ML framework-agnostic toolkits 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 (e.g., JAX (Bradbury et al., 2018), PyTorch Lightning (W. Falcon, 2019)) and therefore the user should choose which framework to use for their local training pipelines. TFF is tightly coupled with TensorFlow and experimentally supports JAX, LEAF also has a dependency on TensorFlow, and Syft 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. Other frameworks are based on Python, with some of them indicating a plan to support Android and iOS (but not embedded platforms) 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 and FedScale come with a number of benchmarks built-in with different datasets. TFF provides libraries for constructing baselines with some 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., 2018).

4 Implementation

Flower has an extensive implementation of FL averaging algorithms, a robust communication stack, and various examples of deploying Flower on real and simulated clients. Due to space constraints, we only focus on some of the implementation details in this section and refer the reader to the Flower GitHub repository for more details.

Communication stack. 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 its 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.

Serialization. Independent of communication stack, Flower clients receive instructions (messages) as raw byte arrays (either via the network or throught other means, for example, inter-process communication), deserialize the instruction, and execute the instruction (e.g., training on local data). The results are then serialized and communicated back to the server. Note that a client communicates with the server through language-independent messages and can thus be implemented in a variety of programming languages, a key property to enable real on-device execution. The user-accessible byte array abstraction makes Flower uniquely serialization-agnostic and enables users to experiment with custom serialization methods, for example, gradient compression or encryption.

Alternative communication stacks. 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 enables the server to support user-provided RPC frameworks and orchestrate workloads across heterogeneous clients, with some connected through gRPC, and others through other RPC frameworks.

ClientProxy. The abstraction that enables communication-agnostic execution is called *ClientProxy*. Each ClientProxy object registered with the ClientManager represents a single client that is available to the server for training or evaluation. Clients which are offline do not have an associated ClientProxy object. All server-side logic (client configuration, receiving results from clients) is built against the ClientProxy abstraction.

One key design decision that makes Flower so flexible is that ClientProxy is an abstract interface, not an implementation. There are different implementations of the ClientProxy interface, for example, GrpcClientProxy. Each implementation encapsulates details on how to communicate with the actual client, for example, to send messages to an actual edge device using gRPC.

Virtual Client Engine (VCE). Resource consumption (CPU, GPU, RAM, VRAM, etc.) is the major bottleneck for large-scale experiments. Even a modestly sized model easily exhausts most systems if kept in memory a million times. The VCE enables large-scale single-machine or multi-machine experiments by executing workloads in a resource-aware fashion that either increases parallelism for better wall-clock time or to enable large-scale experiments on limited hardware resources. It creates a *ClientProxy* for each client, but defers instantiation of the actual client object (including local model and data) until the resources to execute the client-side task (training, evaluation) become available. This avoids having to keep multiple client-side models and datasets in memory at any given point in time.

VCE builds on the Ray (Moritz et al., 2018) framework to schedule the execution of client-side tasks. In case of limited resources, Ray can sequence the execution of client-side computations, thus enabling a much larger scale of experiments on common hardware. 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-cohort settings.

5 Framework Evaluation

In this section we evaluate Flower's capabilities in supporting both research and implementations of real-world FL workloads. Our evaluation focuses on three main aspects:

- **Scalability**: We show that Flower can (a) efficiently make use of available resources in single-machine simulations and (b) run experiments with millions of clients whilst sampling thousands in each training.
- **Heterogeneity**: We show that Flower can be deployed in real, heterogeneous devices commonly found in cross-device scenario and how it can be used to measure system statistics.
- **Realism**: We show through a case study how Flower can throw light on the performance of FL under heterogeneous clients with different computational and network capabilities.
- **Privacy**: Finally, we show how our implementation of Secure Aggregation matches the expected theoretical overhead as expected.

5.1 Large-Scale Experiment

Federated Learning receives most of its power from its ability to leverage data from millions of users. However, selecting large numbers of clients in each training round does not necessarily translate into faster convergence times. In fact, as observed in (McMahan et al., 2017), there is usually an empirical threshold for which if we increase the number of participating clients per round beyond that point, convergence will be slower. By allowing experiments to run at mega-scales, with thousands of active clients per round, Flower gives us the opportunity to empirically find such threshold for any task at hand.

To show this ability, in this series of experiments we use Flower to fine-tune a network on data from 15M users using different numbers of clients per round. More specifically, we fine-tune a Transformer network to correctly predict Amazon book ratings based on text reviews from users.

Experimental Setup. We choose to use Amazon's Book Reviews Dataset (Ni et al., 2019) which contains over 51M reviews from 15M different users. Each review from a given user contains a textual review of a book along with its given rank (1-5). We fine-tune the classifier of a pre-trained DistilBERT model (Sanh et al., 2019) to correctly predict ranks based on textual reviews. For each experiment we fix the number of clients being sampled in each round (from 10 to 1000) and aggregate models using `FedAvg`. We test the aggregated model after each round on a fixed set of 1M clients. Convergence curves are reported in Figure 3 all our experiments were run using two NVIDIA V100 GPUs on a 22-cores of an Intel Xeon Gold 6152 (2.10GHz) CPU.

Results. Figure 3 shows the expected initial speed-up in convergence when selecting 10 to 500 clients per round in each experiment. However, if we decide to sample 1k clients in each round, we notice an increase in convergence time. Intuitively, this behaviour is caused by clients' data having very different distributions; making it difficult for simple Aggregation Strategies such as `FedAvg` to find a suitable set of weights.

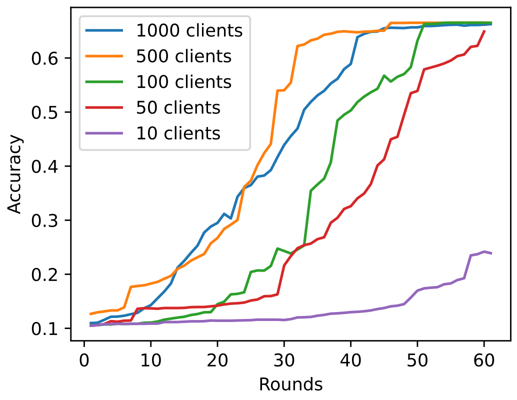

Figure 3. Flower scales to even 15M user experiments. Each curve shows successful convergence of the DistilBERT model under varying amounts of clients per round, with the exception of the two smallest client sizes: 50 and 10.

5.2 Single Machine Experiments

One of our strongest claims in this paper is that Flower can be effectively used in Research. For this to be true, Flower needs to be fast at providing reliable results when experimenting new ideas, e.g. a new aggregation strategy.

In this experiment, we provide a head-to-head comparison in term of training times between Flower and the four main FL frameworks, namely FedScale, TFF, FedJax and the original LEAF, when training with different FL setups.

Experimental Setup. We consider all three FL setups proposed by (Caldas et al., 2018) when training a CNN model to correctly classify characters from the *FEMNIST* dataset. More specifically, we consider the scenarios where the number of clients (c) and local epochs per round change (l) vary. The total number of rounds and total number of clients are kept constant at 2000 and 179, respectively. To allow for a fair comparison, We run all our experiments using eight cores of an Intel Xeon E5-2680 CPU (2.40GHz) equipped with two NVIDIA RTX2080 GPUs and 20GB of RAM.

Results. Figure 4 shows the impact of choosing different FL frameworks for the various tasks. On our first task, when training using three clients per round ($c = 3$) for one local epoch ($l = 1$), FedJax finishes training first (05:18), LEAF finishes second (44:39) followed by TFF (58:29) and Flower (59:19). In this simple case, the overhead of having a multi-task system, like the Virtual Client Engine (VCE), causes Flower to sightly under-perform in comparison to loop-based simulators, like LEAF.

However, the benefits of having a VCE become more evident if we train on more realistic scenarios. When increasing the number of clients per round to 35 while keeping the single local epoch, we notice that Flower (230:18) is still among the fastest frameworks. Since the number of local epochs is still one, most of the overhead comes from loading data and models into memory rather than performing real training, hence the similarity those LEAF and Flower.

The VCE allows us to specify the amount of GPU memory we want to associate with each client, this allows for more efficient data and model loading of different clients on the same GPU, making the overall training considerably faster. In fact, when we substantially increase the amount of work performed by each client to 100 local epochs, while fixing the number of active client to 3, we see a significant saving in training time. In this task Flower outperforms all other. It

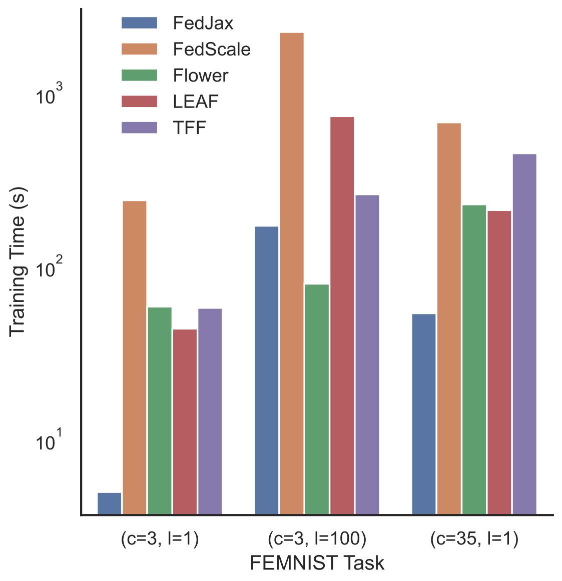

Figure 4. Training times (log scale in second) comparison of different FEMNIST tasks between different FL frameworks.

completes the task in just about 80 minutes, while the second best performing framework (FedJax) takes over twice as long (over 173 minutes).

It is also important to acknowledge the two extreme training times we see in this experiment. FedJax seems to be very efficient when training on few (1) local epochs; however, in scenarios where communication-efficiency is key and larger number of local epochs are required, FedJax performance slightly degrades. FedScale, on the other hands, consistently showed high training times across all training scenarios. We believe this apparent inefficiency to be associated with network overheads that are usually unnecessary in a single-computer simulation.

5.3 Flower enables FL evaluation on real devices

Flower can assist researchers in quantifying the system costs associated with running FL on real devices and to identify bottlenecks in real-world federated training. In this section, we present the results of deploying Flower on six types of heterogeneous real-world mobile and embedded devices, including Java-based Android smartphones and Python-based Nvidia Jetson series devices and Raspberry Pi.

Experiment Setup. We run the Flower server configured with the `FedAvg` strategy and host it on a cloud virtual machine. Python-based Flower clients are implemented for Nvidia Jetson series devices (Jetson Nano, TX2, NX, AGX) and Raspberry Pi, and trained using TensorFlow as the ML framework on each client. On the other hand, Android smartphones 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 in Java. While TFLite is primarily designed for on-device inference, we leverage its capabilities to do on-device model personalization to implement a FL client application (Lite, 2020). The source code for both implementations is available in the Flower repository.

Results. Figure 5 shows the system metrics associated with training a DeepConvLSTM (Singh et al., 2021) model for a human activity recognition task on Python-enabled Jetson and Raspberry Pi devices. We used the RealWorld dataset (Sztyler & Stuckenschmidt, 2016) consisting of time-series data from accelerometer and gyroscope sensors on mobile devices, and partitioned it across 10 mobile clients. The first takeaway from our experiments in that we could deploy Flower clients on these heterogeneous devices, without requiring any modifications in the client-side Flower code. The only consideration was to ensure that a compatible ML framework (e.g., TensorFlow) is installed on each client. Secondly, we show in Figure 5 how FL researchers can deploy and quantify the training time and energy consumption of FL on various heterogeneous devices and processors. Here, the FL training time is aggregated over 40 rounds, and includes the time taken to perform local 10 local epochs of SGD on the client, communicating model parameters between the server and the client, and updating the global model on the server. By comparing the relative energy consumption and training times across various devices, FL researchers can devise more informed client selection policies that can tradeoff between FL convergence time and overall energy consumption. For instance, choosing Jetson Nano-CPU based FL clients over Raspberry Pi clients may increase FL convergence time by 10 minutes, however it reduces the overall energy consumption by almost 60%.

Next, we illustrate how Flower can enable fine-grained profiling of FL on real devices. We deploy Flower on 10 Android clients to train a model with 2 convolutional layers and 3 fully-connected layers (Flower, 2021) on the CIFAR-10 dataset. TensorFlow Lite is used as the training ML framework on the devices. We measure the time taken for various FL operations, such as local SGD training, communication between the server and client, local evaluation on the client, and the overhead due to the Flower framework.

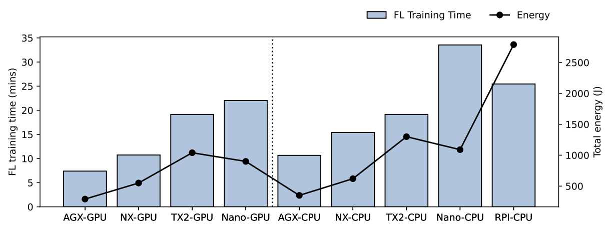

Figure 5. Flower enables quantifying the system performance of FL on mobile and embedded devices. Here we report the training times and energy consumption associated with running FL on CPUs and GPUs of various embedded devices.

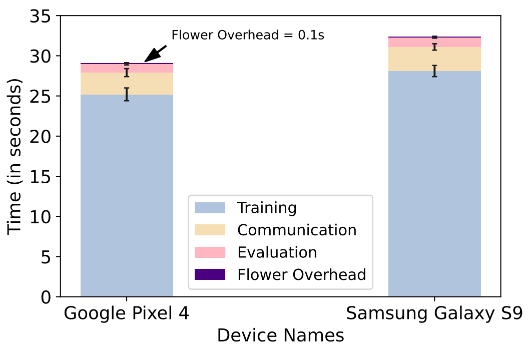

Figure 6. Flower enables fine-grained profiling of FL performance on real devices. The framework overhead is <100ms per round.

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	**CPU (τ = 0)**	**CPU (τ = 2.23)**	**CPU (τ = 1.99)**
Accuracy	0.67	0.67	0.66	0.63
Training time (mins)	80.32	102 (1.27×)	89.15 (1.11×)	80.34 (1.0×)

The overhead includes converting model gradients to GRPC-compatible buffers and vice-versa, to enable communication between Java FL clients and a Python FL server. In Figure 6, we report the mean latency of various FL operations over 40 rounds on two types of Android devices: Google Pixel 4 and Samsung Galaxy S9. We observe that on both devices, local training remains the most time-consuming operation, and that the total system overhead of the Flower framework is less than 100ms per round.

5.4 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 Jetson TX2 as the client device, which has a Pascal GPU and six CPU cores. We train a ResNet18 model on the CIFAR-10 dataset in a federated setting with 10 total Jetson TX2 clients and 40 rounds of training. In Table 3, we observe that if Jetson TX2 CPU clients are used for federated training (local epochs E=10), the FL process would take 1.27× more time to converge as compared to training on Jetson TX2 GPU clients.

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 Flower's on-device training capabilities is that we can accurately measure and assign a *realistic* processor-specific cutoff time for each client. For example, we measure that on average it takes 1.99 minutes to complete a FL round on the TX2 GPU. We then set the same time as a cutoff for CPU clients (τ = 1.99 mins) as shown in Table 3. This ensures that we can obtain faster convergence even in the presence of CPU clients, at the expense of a 4% accuracy drop. With τ = 2.23, a better balance between accuracy and convergence time could be obtained for CPU clients.

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).

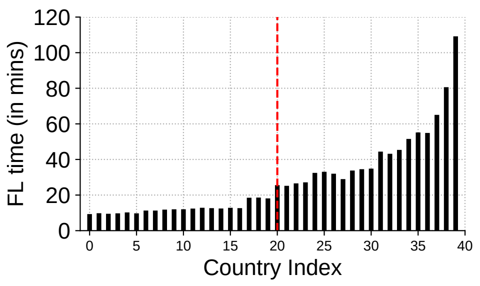

Figure 7. Effect of network heterogeneity in clients on FL training time. Using this quantification, we designed a new client sampling strategy called FedFS (detailed in the Appendix).

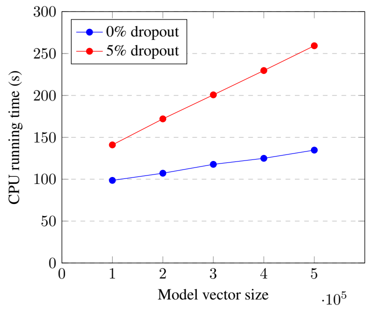

Figure 8. Performance of Secure Aggregation. Running time of server with increasing vector size

The x-axis of Figure 7 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 detailed in the appendix works on this general idea, and reduces the convergence time of FL by up to 30% over the FedAvg random sampling approach.

5.5 Secure Aggregation Overheads

Privacy is one of the cornerstones in FL, which inevitably generates computational overhead during training. In hardware-constrained systems, such as cross-device FL, it is desirable not only to be able to measure such overheads, but also to make sure that security protocols are well implemented and follow the expected protocol described in the original papers. Flower's implementation of Secure Aggregation, named Salvia, is based on the SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020) protocols as described in Section 3.6. To verify that Salvia's behavior matches the expected theoretical complexity, we evaluate its impact on server-side computation and communication overhead with the model vector size and clients dropouts.

Experiment Setup. The FL simulations run on a Linux system with an Intel Xeon E-2136 CPU (3.30GHz), with 256 GB of RAM. In our simulations, all entries of our local vectors are of size 24 bits. We ignore communication latency. Moreover, all dropouts simulated happen after stage 2, i.e. Share Keys Stage. This is because this imposes the most significant overhead as the server not only needs to regenerate dropped-out clients' secrets, but also compute their pairwise masks generated between their neighbours.

For our simulations, the n and t parameters of the t-out-of-n secret-sharing scheme are set to 51 and 26, respectively. These parameters are chosen to reference SecAgg+'s proven correctness and security guarantees, where we can tolerate up to 5% dropouts and 5% corrupted clients with correctness holding with probability $1 - 2^{-20}$ and security holding with probability $1 - 2^{-40}$.

Results. Fixing the number of sampled clients to 100, we plotted CPU running times through aggregating a vector of size 100k entries to aggregating one of size 500k entries in Figure 8. We also measured how the performance would change after client dropouts by repeating the same experiments with a 5% client dropout.

Both the running times and total data transfer of the server increase linearly with the model vector size as the operations involving model vectors are linear to the vectors' sizes, e.g. generating masks, sending vectors. We also note the server's running time increases when there are 5% clients dropping out, as the server has to perform extra computation to calculate all k pairwise masks for each client dropped. Lastly, we observe that the total data transferred of the server remains unchanged with client dropouts as each client only communicates with the server plus exactly k neighbors, regardless of the total number of clients and dropouts. We conclude that all our experimental data matches the expected complexities of SecAgg and SecAgg+.

6 Conclusion

We have presented Flower – a novel framework that is specifically designed to advance FL research by enabling 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 real-world FL systems. Through the provided abstractions and components, researchers can federated existing ML workloads (regardless of the ML framework used) and transition these workloads from large-scale simulation to execution on heterogeneous edge devices. We further evaluate the capabilities of Flower in experiments that target both scale and systems heterogeneity by scaling FL up to 15M clients, providing head-to-head comparison between different FL frameworks for single-computer experiments, 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.

Bell, J. H., Bonawitz, K. A., Gascón, A., Lepoint, T., and Raykova, M. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1253–1269, 2020.

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.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

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

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.

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.

Flower. Model architecture for android devices. https://github.com/adap/flower/blob/main/examples/android/tflite_convertor/convert_to_tflite.py, 2021.

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.

Lai, F., Dai, Y., Zhu, X., and Chowdhury, M. Fedscale: Benchmarking model and system performance of federated learning. *arXiv preprint arXiv:2105.11367*, 2021.

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.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks, 2020.

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.

Lite, T. On-device model personalization. https://blog.tensorflow.org/2019/12/example-on-device-model-personalization.html, 2020.

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.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., and Stoica, I. Ray: A distributed framework for emerging ai applications, 2018.

Ni, J., Li, J., and McAuley, J. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 188–197, 2019.

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., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., and McMahan, H. B. Adaptive federated optimization, 2021.

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.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

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.

Singh, S. P., Sharma, M. K., Lay-Ekuakille, A., Gangwar, D., and Gupta, S. Deep convlstm with self-attention for human activity decoding using wearable sensors. *IEEE Sensors Journal*, 21(6):8575–8582, Mar 2021. ISSN 2379-9153. doi: 10.1109/jsen.2020.3045135. URL http://dx.doi.org/10.1109/JSEN.2020.3045135.

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.

Sztyler, T. and Stuckenschmidt, H. On-body localization of wearable devices: An investigation of position-aware activity recognition. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–9. IEEE Computer Society, 2016. doi: 10.1109/PERCOM.2016.7456521.

W. Falcon, e. a. Pytorch lightning, 2019. URL https://github.com/williamFalcon/pytorch-lightning.

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 2019. PMLR. URL http://proceedings.mlr.press/v97/xie19b.html.

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 Survey on papers

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. Fig. 9 shows the histograms of total number of clients in the FL pools in research papers. The research papers is gathered from Google Scholar that is related to federated learning from last 2 years which consists of total 150 papers in the survey. We excluded papers that are using the framework not available to reproduced the results. As we can see from the histogram, the majority of experiments only use up to 100 total clients, which usually on datasets such as CIFAR10 and ImageNet. There are only 3 papers using the dataset with a total clients pool up to 1 millions, and they are using the Reddit and Sentiment140 dataset from leaf (Caldas et al., 2018).

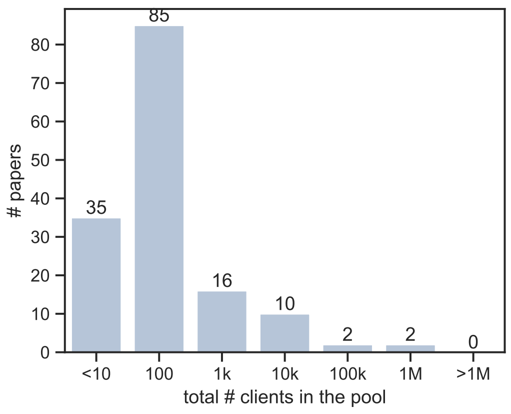

Figure 9. Histograms of the number of total FL clients used in FL research papers in the last two years. A vast majority of papers only use up to 100 clients.

A.2 FedFS Algorithm

We introduce *Federating: Fast and Slow* (*FedFS*) to overcomes 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 $\theta_{k\in\{0,..,K\}}$ 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 $c_k < E\frac{m_k}{B}B$. 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 $r \in \{1, ..., t\}$, the amount of work done by client k during all previous rounds up to the current round t) and $E^r m_k$ (with $r \in \{1, ..., t\}$, 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 $k \in \{1, ..., K\}$), 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 contributions (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 Algorithm 1.

Algorithm 1: FedFS

```
begin Server $T, C, K, \epsilon, r_f, r_s, \Delta_{max}, E, B,$
    initialise $\theta_0$
    for round $t \leftarrow 0, ..., T-1$ do
        $j \leftarrow \max(\lfloor C \cdot K \rfloor, 1)$
        $\mathcal{S}_t \leftarrow$ (sample $j$ distinct indices from $\{1, ..., K\}$
          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
```

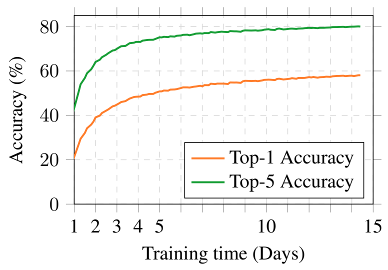

Figure 10. Training time reported in days and accuracies (Top-1 and Top-5) for an ImageNet federated training with Flower.

A.3 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 10 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 10 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.

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.

A.4 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.

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 (Russakovsky et al., 2015) containing $1.2M$ images for training and $50K$ images for testing. A ResNet-18 model is used for federated training this dataset.

High-throughput Simulation of Federated Learning via Resource-Aware Client Placement

Lorenzo Sani*
ls985@cam.ac.uk
University of Cambridge

Pedro Porto Buarque de Gusmão*
pp524@cam.ac.uk
University of Cambridge

Alex Iacob*
aai30@cam.ac.uk
University of Cambridge

Wanru Zhao
wz341@cam.ac.uk
University of Cambridge

Xinchi Qiu
xq227@cam.ac.uk
University of Cambridge

Yan Gao
yg381@cam.ac.uk
University of Cambridge

Javier Fernandez-Marques
jafermarq@gmail.com
Samsung AI

Nicholas Donald Lane
ndl32@cam.ac.uk
University of Cambridge

ABSTRACT

Federated Learning (FL) is the privacy-preserving machine learning paradigm which collaboratively trains a model across millions of devices. Simulated environments are fundamental to large-scale FL research, allowing researchers to quickly test new ideas to solve system and statistical heterogeneity issues. This work proposes *Pollen*, a novel resource-aware system capable of speeding up FL simulations by efficiently placing clients across distributed and heterogeneous hardware. We propose minimising server-GPU communication and using an efficient client placement policy based on the inherent trade-offs of FL client placement on heterogeneous GPUs. These trade-offs are explored experimentally.

This exploration has been conducted via relevant baselines on three popular FL tasks: image classification, speech recognition and text generation. We compare *Pollen* to existing ad-hoc FL frameworks, such as Flower, Flute and FedScale, and show performance gains of 50% to 400%.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; *Neural networks*; **Parallel computing methodologies**; **Parallel algorithms**; *Massively parallel algorithms*; **Simulation environments**; • **Computer systems organization**;

KEYWORDS

federated learning, simulation, scalability, deep learning, resource management

*These authors contributed equally to this research.

1 INTRODUCTION

Machine learning (ML) is becoming increasingly viable on constrained hardware with low memory such as smartphones, wearables, and general IoT devices. Techniques originally developed for training models on smartphones with hundreds of MB of memory [10] can now run on micro-controllers with 256kB [19]. Federated learning (FL) was introduced by McMahan et al. [21] to allow training on thousands to millions of such edge clients in a distributed manner while avoiding the privacy and network costs of communicating their sensitive data in a centralised fashion.

While Federated learning allows edge devices to collaborate effectively, Kairouz et al. [12, sec. 3.1] indicate that it brings unique challenges relating to clients' different hardware and data distributions. To effectively model and address such challenges, researchers must be able to run large-scale Federated Learning simulations efficiently on their hardware.

Unlike centralised ML, dominated by long-running execution of large jobs, simulating federated learning relies on the repeated execution of small clients corresponding to the resource-constrained devices it intends to emulate. Since such clients cannot keep GPU utilisation high by themselves, scheduling many of them to run both on the same GPU and across GPUs is beneficial. Crucially, clients can vary in dataset size by orders of magnitude. Independently on the client selection procedure, existing FL frameworks [2, 7, 16] treat all clients equally during simulations and can often end up with straggling GPUs. A proper *placement* of clients on GPUs could significantly reduce training time and allow more extensive simulations. Unfortunately, such intelligent resource-based *placement* is impossible in existing FL frameworks as they rely on a *pull-based* system where workers sequentially sample clients from a server queue.

In this work, we propose *Pollen*, an adaptive *push-based* method for client placement in simulated FL, that is compatible with diverse client selection procedures, and show that it significantly improves FL training times. Instead of building an entirely new framework, we chose to modify Flower [2] due to its flexibility. We showcase the design of our system and its effectiveness and analyse the factors which decide the training time of the simulation for a given placement policy. Our contribution is *threefold:*

(1) We experimentally show a proportional but nonlinear relationship between the size of a clients' dataset and their training time across multiple workloads and GPU types. We argue that this requires intelligent client placement to optimise training time when combined with the skewness of standard FL datasets.
(2) We build *Pollen*, a client placement system to effectively partition clients amongst potentially heterogeneous GPUs using several placement strategies. Then, we compare the effectiveness of such strategies. Our results show that for heterogeneous GPU environments, a learning-based policy outperforms others by up to 81% as it can accurately learn to predict training time regardless of system configuration.
(3) Finally, we show that *push-based* client placement can significantly improve training time over previous *pull-based* systems. *Pollen* achieves a 50% to 378% reduction in training time across various datasets when training 10 000 clients split evenly across 100 rounds with heterogeneous GPUs. Moreover, for homogeneous GPUs, our method decreases communication costs enough to obtain a 200% to 400% improvement over the next-fastest system.

Our method allows FL researchers to run more extensive, faster, and scalable simulations. These improvements permit rapid prototyping and development of algorithms for production settings involving millions of edge devices.

2 SIMULATING FEDERATED LEARNING

We now introduce federated learning and describe the main characteristics of the standard simulation solutions adopted by popular federated learning frameworks. We then justify our proposed system *Pollen* based on the observed limitations of previous systems.

2.1 Federated Learning

Federated Learning is concerned with training ML models in a distributed fashion while minimising communication costs and maintaining private data on-device. In its most popular cross-device version [12], it takes the form of synchronised training where many clients pool their resources to train a model collaboratively. Such devices can differ significantly both in terms of their local data and in terms of available hardware. For example, a group of devices for human activity recognition could be composed of smartphones containing gyroscopes and accelerometers [25, 26], surveillance video cameras [15], and passive sensing devices using Radio-Frequency data [13]. Clients in these diverse categories would all have highly divergent data modalities, dataset sizes, computational power, network speed and training availability. Human activity recognition data can be sensitive and often needs to stay private to the point of origin, thus requiring a federated approach.

Federated Learning algorithms, like Federated Averaging [21] for cross-device FL, maintain data privacy via a client-server training design synchronised across *rounds*. The server controls the training and holds the federated model. It sends the model to each client at the start of the round, where it is trained on private data using Stochastic Gradient Descent (SGD). Then, the clients return the models to the server, aggregating them to create a new federated model for the next round.

2.2 Framework Simulation Engines

The simulation engines of FL frameworks aim to enable experiments in a constrained environment where we may not have enough resources to virtualise all the clients in the FL setting or all the clients in the sampled cohort for each round. We can better understand this scenario by representing each client's training as a job to schedule. Each of these jobs needs to allocate resources to account for a complete copy of the model to train, the dataset for the client, including the preprocessing, and the instructions for the training. The needs of a single client are usually relatively small; as such, it is possible to fit many of them into a single GPU.

Given this setting, FL simulation engines try orchestrating the training using a server-workers paradigm. The server orchestrates the simulated FL training by serving clients to workers and aggregates the results after every round of training. The workers are responsible for training clients individually and sending the trained models to the server once the training is complete. Ad-hoc FL frameworks, such as Flute [7], FedScale [16] and Flower [2], rely on this paradigm.

It is worth highlighting that workers in both FedScale and Flute are statically allocated to their GPU. On the other hand, Flower's Virtual Client Engine, developed by Beutel et al. [2] and based on Ray [22], can dynamically move workers between GPUs during the FL training. However, the control over this dynamic allocation is limited. In all these frameworks, the server serves clients to workers using a pull-based queuing system involving many communication steps between the server and workers. The execution of an FL round on this system can be summarised as follows.

(1) At the start of a round, the server samples a subset of participating clients. This subset defines the cohort of clients to be trained in that round. The cohort is kept in a synchronised queue, allowing workers to read clients sequentially.
(2) Each worker reads from the queue, extracts the first client, and starts the training. Different workers cannot access the same element of the synchronised list.
(3) Once a worker finishes training a client, it pings the server to notify that the client's training is completed.
(4) When the server receives this message, it replies to the worker when it can receive the training results.
(5) The worker finally sends the results to the server that will store them or partially aggregate them, depending on the experiment's configuration.

The number of workers controls the concurrency of this embarrassingly parallel simulation. Hence, it is beneficial to increase the number of workers up until the resources of the system are saturated, or gains from concurrency cease.

2.3 Limitations of Current Systems

The limitations of pull-based queuing systems are multifold. Critically, the workers of a specific GPU cannot choose which client to train. This underlying lack of control can limit the simulator's options when attempting to train specific clients on specific GPUs. For example, when disproportionately large clients are selected, balancing client training time across GPUs and avoiding stragglers is impossible.

Another general limitation is that the communication involved may take a significant amount of time relative to the training time of most clients. Such communication bottlenecks are significant for settings with multiple machines (nodes) that need a network connection to communicate with each other. In this work, *multinode* refers to hardware configurations with GPUs distributed amongst separate machines.

We now present the specific limitations of each framework addressed in our paper.

Flute, introduced by Dimitriadis et al. [7], is optimised to use the *nccl* backend of PyTorch Distributed [17] when running on GPUs and the *gloo* backend for CPU training. It can only run a single worker per GPU and, because it cannot intermix GPU and CPU training, it requires an entire GPU to hold the parameter server that handles aggregation during FL simulation. This can be wasteful as aggregation is usually not a compute-intensive operation. These issues cannot be easily addressed as the codebase has highly coupled components dependent on this design.

FedScale, introduced by Lai et al. [16], depends on unreliable configurations of gRPC, which is felt across the many communication steps necessary for the pull-based design. As such, longer rounds may cause crashes as clients appear to either disconnect or time out. Furthermore, despite being able to place multiple workers on the same GPU, later experiments show minimal benefits when increasing either the number of workers or GPUs. Each worker is also tasked with loading the entire dataset, even if they are on the same node as other workers and can share memory. Finally, similarly to Flute, the codebase coupling level makes refactoring difficult.

Flower, introduced by Beutel et al. [2], depends on Ray [22] as its simulation engine. Ray may cause out of memory (OOM) in a multi-GPU configuration as workers do not fully deallocate memory from previously used GPUs. Careful configuration of parameters helps avoid OOM at the cost of slowing down the overall training. However, unlike Flute and FedScale, the codebase is highly modular, allowing us to implement our new simulation engine on top of existing components.

General limitations of GPU scheduling for ML tasks are also worth mentioning. Job scheduling on a GPU cluster for traditional ML tasks is a well-studied problem [8, 29, 30]. For example, Gandiva [29] schedules large jobs on such a cluster by profiling the rate at which they process mini-batches. At the same time, CODA [30] attempts to balance the CPU assignment of GPU jobs to optimise data loading. However, both rely on the assumption that ML tasks are highly repetitive and run long enough to be effectively optimised. As we will show in Section 3, client dataset sizes in FL are much smaller than a typical ML job and more difficult to profile. Furthermore, client dataset sizes are highly skewed, making round durations unpredictable. Together, these two considerations make the heuristics of existing GPU scheduling systems insufficient for FL. This insufficiency drives us to propose an FL-specific solution rather than plugging in an existing one.

3 HETEROGENEITY AND CLIENT PLACEMENT

We now describe the practical difficulties of deciding which GPU a client should be placed on for training. For these, we identify two causes in the form of client heterogeneity and hardware heterogeneity.

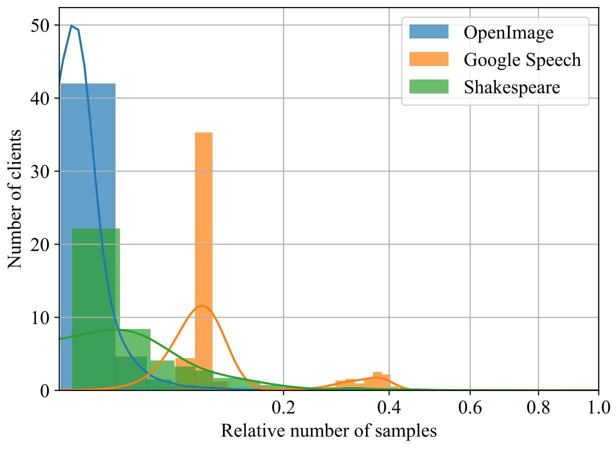

Figure 1: Dataset size distribution over clients for OpenImage [14], Google Speech [28], and Shakespeare [4]. Natural non-uniformity in data distributions will lead to different training times. A non-linear scale was chosen for the x-axis.

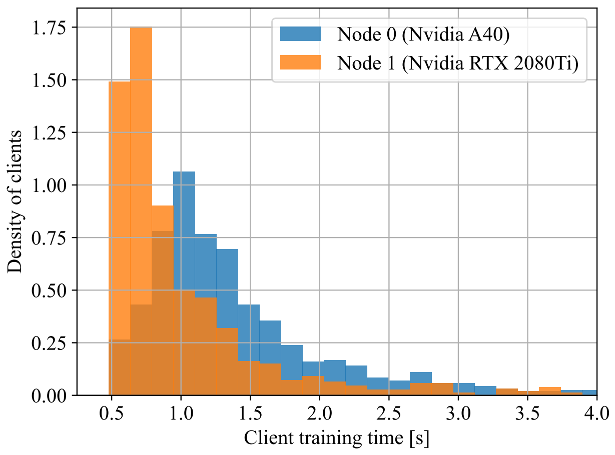

Figure 2: Training times for two GPUs running on different client dataset sizes. The GPUs are running the same clients.

3.1 Client Heterogeneity

In large-scale cross-device FL, clients collect or produce data at vastly different rates [18]. Efficient simulations should reflect this fact and account for the different training times that simulated clients will take. While some frameworks try to circumvent this issue by fixing the number of steps each client will train for [16], this assumes that the dataset distribution is not highly skewed and leads to two issues.

First, it limits the contribution of clients having large datasets. Second, clients having small datasets are forced to reuse their data (increase in local epochs). Consequently, we refrain from fixing a constant number of training steps throughout our experiments and argue that this is not a reasonable assumption to be made at a framework level. Fig. 1 shows the distributions of samples for three different naturally-partitioned datasets commonly used in FL simulations. FL datasets are usually long-tailed and skewed, leading to the abovementioned issues.

We argue that the dissimilarity in dataset distributions makes it necessary to design dynamically adaptive placement strategies capable of accommodating different datasets. Additionally, the various pre-processing pipelines such datasets use for mini-batches reinforce this requirement. For example, image datasets may require samples to be loaded from disk and transformed, while language datasets may store features in memory. We further argue that the internal skewness of a dataset requires intelligent placement of clients on devices even for the simplest case of *homogeneous* GPU configurations. For example, in extreme situations, one device may train clients with orders of magnitude more batches than another.

3.2 Hardware Heterogeneity

Besides client heterogeneity, it is necessary to consider the different GPUs used in FL simulations. As previously mentioned, individual client workloads are smaller than traditional ML tasks and are highly parallelisable, allowing them to be trained across various machines with little effort. However, using heterogeneous GPUs may result in workers finishing processing their clients at very different times, leading to unnecessarily long experiments.

Roughly speaking, training a single client will depend on *data loading* and *actual training*. Data loading and preprocessing generally happen on the CPU. As the number of concurrent CPU jobs increases beyond the number of CPU cores, the variability of the data loading time grows proportionally to that of the scheduling. This can act as a bottleneck when increasing the number of workers available for processing.

The actual training of a client usually happens inside the GPU, and the time it takes is affected by the client's local dataset size. For large clients, the training time is approximately determined by the average speed at which the GPU can process each of their batches. However, for small clients, the startup times are a more significant section of the total time the clients spend executing, which causes increased variability in total training time. Furthermore, we observe some variability even for huge clients, which should be the least affected by startup concerns.

Figure 2 shows the training time distribution of two Nvidia GPUs running the same population of clients with different dataset sizes. As can be observed from the figure, the two GPUs perform very differently from one another. Therefore, allocating less work to the slower GPU is necessary to optimise training time by having them finish simultaneously.

4 POLLEN DESIGN

This section describes the key ideas and components used in our proposed solution. An overview of the complete system, dubbed *Pollen*, can be seen in Fig. 3.

4.1 Placement Strategy

Following our investigation in Section 2, we propose an alternative client placement system that addresses the issues reported in Section 2.3 regarding the limitations of FL frameworks. In our approach, instead of having workers *requesting* clients from the server, the system follows a *placement strategy* to partition client workloads across workers and perform a *push-based* allocation. This method allows us to reduce the number of communication steps between the server and nodes and to assign sets of clients to appropriate GPUs.

It is worth mentioning that our placement method acts on the underlying simulation layer of the FL framework. It is an independent procedure from *client selection* and is not affected by the sampling procedure used to generate the clients for a specific round nor any other algorithmic properties. It can be easily extended to use other sampling techniques such as FedCS [23], Power-of-Choice [6] and DivFL [1].

4.2 Resource Allocator:

Clients having different amounts of data and being trained on heterogeneous GPUs will produce disparities in workers' execution times. A solution to this is to be able to associate client workloads with appropriate GPUs.

At the beginning of the FL simulation, the *resource allocator* module receives information from all training nodes regarding their available hardware, e.g., the number of CPU cores and the number and types of GPUs. This information is used to allocate resources to workers, following user-defined constraints, such as the maximum number of workers per GPU, which can be estimated by profiling a single inference step for each GPU type contained in the node.

4.3 Partial Aggregation:

When using associative aggregation strategies, such as FedAvg [21], the system can benefit from partially aggregating results within a worker before sending the partial result for a last aggregation on the server.

In this approach, the worker keeps both a partially aggregate model θ_k^p and a total number of processed data samples N_k after having trained its k-th client, as seen in Eq. (1).

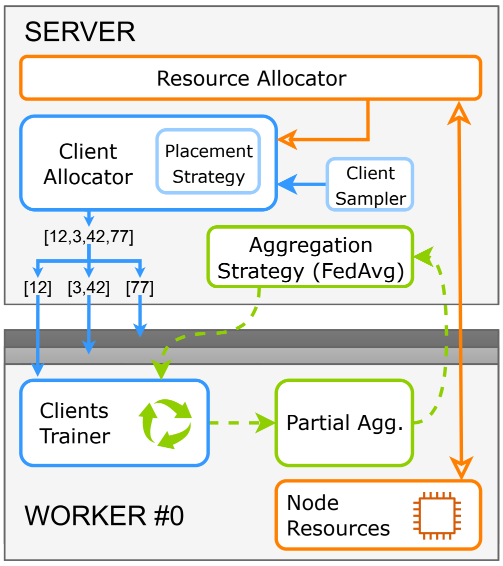

Figure 3: Diagram describing *Pollen* with its relevant elements in both server and worker. The colour code has been used to distinguish between components related to clients (blue), models (green), and hardware (orange).

$$\theta_{k+1}^{p} = \frac{\theta_k^p \times N_k + \theta_{k+1} \times n_{k+1}}{N_{k+1}} \tag{1}$$

$$N_{k+1} = N_k + n_{k+1} \tag{2}$$

Once the *worker* has completed training its list of clients, it will send both the partially aggregated model and the total sum of the samples for the final aggregation.

4.4 Client Allocator:

We follow the concept from Section 2 and define a *worker* as an entity capable of sequentially training lists of clients.

As FL clients can only use small batch sizes and models when training on edge devices, packing many workers and oversubscribing GPUs has proven beneficial in realistic FL simulations, as seen in Fig. 4.

The *client allocator* associates clients with individual workers on specific GPUs. The server samples a list of participating clients at the beginning of a round. Then, it passes this module, which uses a pre-defined *client placement policy*, discussed in Section 5, to determine which worker will train which client and in what order.

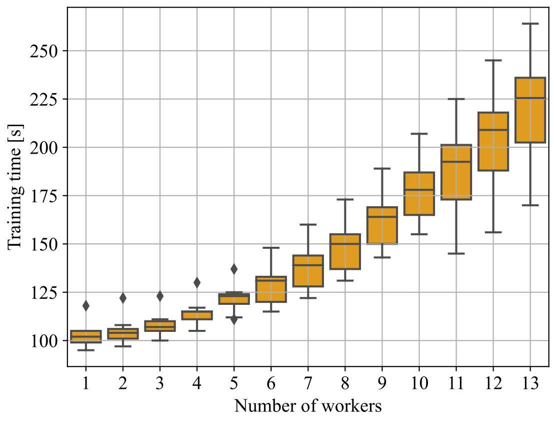

Figure 4: The distribution of clients' training time for different values of concurrent workers on the same GPU. Every worker has the same load: the same list of 100 clients. For example, with 5 workers, the GPU trains 500 clients using 5 concurrent processes.

5 CLIENT PLACEMENT STRATEGIES

This section discusses the strategies for placing clients on GPUs that we have explored in this work.

Since our design allows the simulation to have only one communication step from the server to the workers, our exploration focused on distributing the list of N randomly sampled clients across workers in one step. For each round, all these strategies receive the list of integer clients' IDs to be trained, and they return a list of clients' IDs for each worker indicating which clients it should train. Since the exploration space for determining the best distribution procedure is very broad, we rely on FL features common to most experiments.

Naı̈ve round-robin (RR): This strategy represents the starting point of our investigation since it naively splits the list of clients in k uniformly populated lists, where k is the number of workers. In particular, the first sampled client will be assigned to the first worker and the second client to the second worker, etc. If $\frac{N}{k}$ is not an integer, the remainder is distributed across the first workers.

Batch-sorted round-robin (SRR): The first information we used in this study was the number of batches m each client has. As discussed above, the number of batches is a proxy for the training time in epochs-based FL training. Before naively splitting the list as before, this strategy orders the clients by m from top to bottom. The intuition behind this procedure is that different workers will likely train the biggest clients.

Batch-Uniform distribution (BU): A step forward to the previous strategy is to use the same information while changing the distribution procedure. For homogeneous GPUs, we want the load across workers to be balanced. To achieve this, the strategy loops over the N clients after ordering them by m from top to bottom and assigns the current client to the worker whose load is lower. The load is estimated by summing the number of batches of all the clients assigned to the worker. It has to be noted that the first k clients of the list are assigned the same way as the previous strategy.

Learning-based time prediction (LB): In settings with heterogeneous GPUs, the proxy for the training time given by m may not be sufficient, as shown in previous sections. Our learning-based strategy predicts the training time for each GPU. The first FL round will use the naı̈ve round-robin strategy to collect data about client training time from all available workers. Starting from the second round, the strategy builds one dataset for each GPU composed of tuples $(client\ training\ time,\ m)$ from previous rounds.

Then, for each dataset, the strategy fits the data points to the function in Eq. (3), where y represents the client training time, and x is the number of batches the client has.

$$y = ax + b\log(cx) + d \tag{3}$$

The parameters derived from the fitting are then used for predicting the training time of the clients sampled in the current round. We chose Eq. (3) in order to match the skewed training time distribution we observed empirically in Fig. 1, this choice is further discussed at the end of Section 5.1. The strategy sorts the workers by GPU type, from the fastest to the slowest, using the predicted training time of the biggest client in the current cohort. Finally, the strategy carries on the placement of clients by balancing the load between workers, similar to the batch uniform strategy. Clients are ordered by m from top to bottom and assigned to the worker whose load is lower. Here, the load is estimated by summing the predicted training time of all the clients assigned to the worker.

5.1 Efficient Client Placement

In this work, we try to answer the question of which placement strategy is better to choose in which context. More importantly, we argue that discussing placement strategies in simulating FL is necessary. The research on large-scale [3, 5, 27] FL often relies on simulating FL settings with large cohorts (in the range $[10^2, 10^4]$) of clients over a relatively limited amount of hardware resources for thousands of rounds. Reducing the latency of experiments as much as possible while exploiting the hardware resources at their best is fundamental. Regardless of the chosen placement strategy, we expect any one-step communication method to outperform

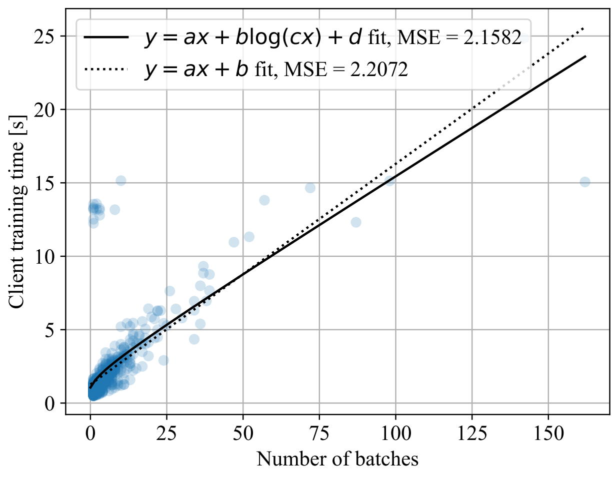

Figure 5: Clients' training times are plotted against their number of batches. The fitting lines of the linear function and the proposed function are shown alongside The Mean Squared Errors (MSE) derived from the fitting procedure.

the queue design in previously mentioned frameworks. However, we will show that an optimal placement can improve the simulation speed by up to 81% compared to the slowest strategy.

Different placement strategies will result in different distributions of clients to train across workers. However, this does not hold for the degenerate placements in which the number of clients trained per round equals the number of workers, and the workers are allocated on the same GPU and node. The Shakespeare baseline is the only one we have explored that can fall into this degenerate case. This is because the training procedure plans to train 10 clients per round, which can easily be fitted into a single GPU. Excluding the degenerate placement, the non-trivial metric that is impacted by different placement strategies is what we call "*timedelta workers*". We define "*timedelta workers*" as the time difference between the timestamp at which the fastest worker finishes their job and the timestamp at which the slowest worker finishes their job, where we intend the job to be the training of all the clients in the received list. When using the RR strategy, we observe that the distribution across rounds of "*timedelta workers*" is peaked at a time at least one order of magnitude greater than the training time of the smallest client in that round. We could have reduced the training time by moving the smallest client from the slowest to the fastest worker. Thus, we assume that "*timedelta workers*" approximates the time wasted due to clients' misplacement.

The impact of the placement strategy on the training time depends on the GPUs we are dealing with. We can distinguish between *homogeneous settings*, in which each worker executes on the same GPU type at the same speed, and *heterogeneous settings*, in which groups of workers are allocated on different GPU types having different speeds.

In *homogeneous settings*, we expect to observe that the ratio between the number of clients per round and the number of workers drives the difference in performance. As this ratio increases, we expect to observe similar performance across strategies. This expectation is motivated by the fact that the noise introduced by the random sampling of clients will likely balance the load over workers naturally. When this ratio is close to 1, only sampled cohorts of clients whose number of large-size clients is different from the number of workers can lead to different performance. In this case, the performance gap appears because big clients cannot be evenly distributed across workers. However, we expect the BU strategy to slightly outperform the others since it can balance the load over workers with the minimum overhead.

The scenario in which different strategies will significantly differ in performance is the *heterogeneous setting* in which clients of the same size are not trained in the same amount of time by different workers. For these settings, the LB strategy will outperform BU because of its ability to discriminate between workers executing on different GPU types. The difference in performance will mostly depend on how heterogeneous the hardware settings are. In particular, the gap between different hardware in training small clients, prevalent in FL datasets, will have the most critical impact.

The LB placement strategy strongly relies on the performance of the fitting procedure over clients' training time from previous rounds. It is worth discussing the two main ingredients upon which this strategy arranges the clients' placement: the data collected for previous rounds and the fitting function. We chose to keep all the data from previously trained rounds. In general, the robustness of curve fitting is proportional to the number of data points; conversely, its duration is proportional to the number of data points. We observe that during the last round, when the data points are 9900, the LB strategy takes an amount of time on the same order of magnitude as the RR strategy, tens of seconds. For extended experiments where the number of data points to fit could be much more significant, it is reasonable to define a time window for deleting older data. Second, our fitting function has been chosen for its mathematical properties. The logarithm combined with a linear term ensures that the fitted function never predicts negative values despite the wide cloud of data points produced by small clients, as can be observed in Fig. 5. Since small clients have greater training time variance, many may take longer than their slightly bigger counterparts. This behaviour may enforce negative slopes in the fitted curve, especially if polynomial. The logarithmic term makes the function more robust to this situation, while the linear term ensures that the bigger clients are predicted to take longer to train. *As such, the chosen function allows us to avoid ever predicting a negative time for a client at the cost of overestimating the duration of small clients.*

6 EXPERIMENTAL DESIGN

This section describes a series of experiments meant to showcase our method's superiority and validate it.

6.1 Federated Learning Tasks

We use three representative FL tasks throughout this work to showcase our method. These are characterised by having clients with long tail distributions over the number of samples and workloads. For all tasks, we exclude clients with less than one batch of training data.

Image Classification: The goal of this task is to collaboratively train a ShuffleNetV2 [20] network to correctly classify images amongst 596 classes. For this, we use a federated version of the original OpenImage [14] dataset as implemented by FedScale. This dataset contains 1.6×10^6 images partitioned across 13 771 clients. We use a batch size of 20 samples.

Speech Recognition: In this task, we use the Google Speech Commands dataset [28] to collaboratively train a ReseNet-34 [11] to classify audio samples amongst a set of 35 predefined spoken words. The dataset contains a collection of $157K$ one-second-long clips naturally partitioned according to their 2168 speakers. We use a batch size of 20 samples.

Text Generation: We use the Shakespeare dataset, as implemented in TensorFlow Federated [9], to train a two-cell LSTM-based language model as defined in [4] The dataset comprises sentences extracted from *The Complete Works of William Shakespeare* and grouped into 648 fictional characters. We follow the LEAF experimental configuration and use a batch size of 4 samples.

6.2 Hardware Configuration

We consider two hardware configurations that reflect common research centres, namely *single-node* and *multi-node*. We also further distinguish between simulations using *homogeneous* and *heterogeneous* GPUs. As previously indicated, all the FedScale, Google Speech, and Shakespeare experiments use a fixed batch size. This makes differences in worker VRAM attributable only to model size and input data shape.

Single-node: Our single-node experiments are all run on *node 0* containing Nvidia A40 GPUs and an Intel (R) Xeon (R) Gold 6152 with 88 cores. For OpenImage, the A40s are filled with 13 workers each, given the size of ShuffleNetV2 [20] and the input size. For Google Speech, we use only 4 workers per GPU due to data loading requirements, while for Shakespeare, we use 10 workers total to match the number of clients. Each A40 is paired with 11 CPU cores out of the 88 mentioned above. We run all our experiments with *Two homogeneous GPUs* on this node using two A40s with 22 CPU cores available.

Multi-node: Our multi-node experiments are run on a combination of the aforementioned *node 0* containing A40s and *node 1. Node 1* contains Nvidia RTX 2080 Ti GPUs and an Intel(R) Xeon(R) CPU E5-2680 v4 containing 56 cores. For OpenImage, we use 4 workers per 2080 due to VRAM constraints. The worker setup for Google Speech and Shakespeare is the same as on *node 0.* Each 2080 is paired with 8 CPU cores out of the 56 mentioned above. Our experiments for heterogeneous hardware shall use one A40 from *node 0* paired with 1-4 Nvidia 2080s. We only use more than one 2080 in scenarios where we want to observe how balancing the number of workers across GPU types affects performance. After the workers have completed their partial aggregation on the *CPU* of their respective node, all final aggregation steps happen on *node 0* on the *CPU.*

6.3 Framework Benchmark

The fundamental contribution of the design of our system is to overcome the pull-based queuing system to allow for fast FL simulation. In this experiment, we aim to assess the speed of our solution, whatever the placement strategies adopted by the client allocator. We compare to the other frameworks by measuring the throughput of the simulation since this describes the system's speed as the number of clients the system can train per second. It is calculated by dividing each round completion time by the number of clients trained. We perform the three tasks under different homogeneous hardware settings having 1, 2 or 4 homogeneous GPUs. For homogeneous hardware, different placement strategies perform similarly. Thus, the performance is mainly impacted by the system design.

6.4 Policies Benchmark

This work proposes different placement strategies to use in the simulation. In this experiment, we aim to answer the question of how the choice of strategy impacts the performance of the simulation. We measure the throughput of the FL simulation and compare different strategies against each other. We are also interested in identifying how effective the strategies are in balancing the load between the workers. To this aim, we measure the time difference between the first and last workers to finish processing their clients and compare the strategies against each other.

The experiment is designed to be executed in the most challenging hardware setting, the heterogeneous one with two nodes having different GPU types. First, we train all the tasks in the setting with one GPU per type for two GPUs. We keep the number of clients per round to 100, except for Shakespeare, where we set 10 clients for 100 rounds. Here, we compare the throughput of the experiments using different strategies against the fastest other framework. Second, we train the Image Classification task in different heterogeneous hardware settings, increasing the number of GPUs belonging to the type that can host the lower number of workers. This second set of experiments uses the two previous nodes, one always having one GPU and the other having 1, 2, 3, and then 4 GPUs.

These settings reflect a typical situation a researcher could face: the available GPUs with more VRAM are few, but more GPUs with less VRAM are available.

6.5 System Scalability

Our main contribution in this paper is to allow efficient large-scale simulations at the scale of real-world applications. In this experiment, we aim to measure how our proposed method compares to other Placement Policies as we *increase the average number of clients per GPU.* The first set of experiments considers the homogeneous scenario of four GPUs of the same type on the same node. We fix the number of workers because the hardware is fixed, and we progressively increase the number of clients per round while keeping constant the overall number of clients trained. In order to maintain the total number of trained clients to 10 000, as it is in other experiments in this work, we choose the values for the number of clients per round to be 100, 200, 400, 625, 1000 training respectively for 100, 50, 25, 16, 10 rounds. The workers receive longer lists of clients to train each round, while the total work is always the same. We measure the system's throughput for all the proposed client policies and then compare them against the fastest framework.

A second set of experiments considers the most constrained and heterogeneous scenario we had: two nodes, each of them having on GPU of a different type. Similarly, we played with the number of clients each worker has to train in each round while keeping the total amount of work constant. For each round, we measure the time difference between the first and last worker to finish processing their clients and the throughput.

We run the above experiments on the Image Classification task with the largest client population. This setting reflects a common FL scenario where researchers want to investigate the benefits of sampling larger fleets of devices in one round while still constrained by the available hardware.

7 EVALUATION

In this section, we describe the results of the experiments proposed in Section 6. We begin presenting the comparison of the performance of *Pollen* against other frameworks. We continue focusing on the differences between the strategy proposed. Furthermore, we extend the investigation of the strategies by evaluating them at scale. Finally, a discussion about the proposed client placement's limitations is presented.

7.1 Comparison between frameworks

Pollen proves to deliver faster FL simulations compared to Flower (version 1.1) [2] , Flute (commit@0e8762b) [7], and FedScale (commit@9bfc029a3c) [16] in every setting we have tested. This consistency demonstrates the superiority of a push-based allocation of clients across workers. The results of the throughput of the simulation in a more accessible homogeneous setting put *Pollen* on top of the classification since it outperforms all the other frameworks. The example in Fig. 6 shows that, when using two GPUs of the same type, namely Nvidia A40, on a single node, *Pollen* has the highest throughput for all the tasks. Compared to the fastest framework, the speed-up obtained is 3x on the Text Generation task, about 2x on the Image Classification task, and 5x on the Speech Recognition task. It is noted that the performance of all the tasks is varied across the other frameworks except for the Speech Recognition one, in which they perform similarly. We highlight that in every comparison in which the number of clients per round is fixed, the absolute time of the experiment is inversely proportional to the reported throughput. In addition, since the Image Classification task has the lowest speed-up factor, we assume that to be the task using which further comparisons will be fairer.

Implications: *Pollen* outperforms all previous frameworks and can provide a great boost to the scalability of FL simulations thus allowing better FL methods to be developed for production systems.

7.2 The Impact of Client Placement

Given that our design outperforms other frameworks in homogeneous settings, we evaluate the impact the client placement strategies in heterogeneous settings. Two nodes were available in the setting, with one Nvidia A40 and one Nvidia RTX 2080 Ti, respectively. The results regarding the throughput are shown in Table 1. We can assume that different strategies deliver the same performance for the Text Generation task because distributing 10 clients across 10 workers is trivial. The Image Classification task shows the biggest variation across strategies, where the LB strategy of *Pollen* outperforms the others. We note that the RR strategy

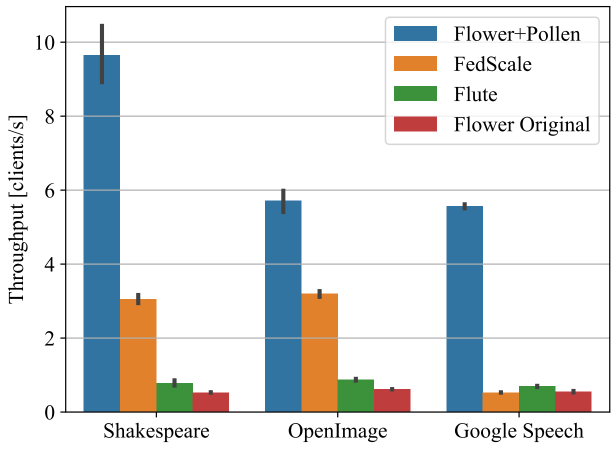

Figure 6: The throughput (the greater, the better) of our client placement systems implemented in Flower compared with FedScale, Flute and standard Flower. Using resource-aware client placement, *Pollen* outperforms the fastest other framework by a factor 3x-5x, depending on the task. The values reported for our contribution have been chosen for the best-performing strategy that tends to be LB.

of *Pollen* underperforms other strategies when the placement is not trivial, presenting the most significant gap in the Image Classification task. We argue that the peculiarity of the Speech Recognition dataset causes different strategies to have more minor variations in performance. Furthermore, we highlight the general superiority of our method against the fastest framework in this challenging heterogeneous scenario.

We gradually increased the number of available GPUs for the second set of experiments to evaluate the impact of different placement strategies. In this experiment, we used the different strategy of *Pollen*. We significantly increased the number of the GPU type that can host the lower amount of workers, namely the Nvidia RTX 2080 Ti. As such, four settings are compared in this multi-node scenario: $(1, 1)$, $(1, 2)$, $(1, 3)$, $(1, 4)$, where the first number refers to the available Nvidia A40s in the first node and the second to the available Nvidia RTX 2080 Ti in the second node. The plot in Fig. 7 shows the throughput of different strategies over the Image Classification task. It is worth noting that the number of clients per round has been kept constant during this particular experiment. In this way, what is changing is the *density* of clients across workers, as having more GPUs available means having more workers. The results demonstrate that the gain produced by using the LB strategy degrades as the density of clients across workers decreases. This degradation is reflected in both the metrics taken into consideration, namely the difference in training time between the fastest and slowest workers (Table 2) and the throughput (Fig. 7).

Implications: Intelligent client placement is highly beneficial for hardware configurations containing multiple GPU types. Given the difficulty of profiling and configuring a large-scale cluster with heterogeneous GPUs, the automatic nature of workload distribution in *Pollen* provides a great advantage.

Table 1: The throughput (the greater, the better) measured in clients per second for three tasks. The proposed *Pollen*'s strategies are compared against FedScale in the most heterogeneous setting.

Datasets	FedScale	LB	BU	SRR	RR
Google Speech	3.6±0.5	4.7±0.4	**5.4±0.2**	5.0±0.2	5.2±0.3
OpenImage	3.6±0.5	**6±1**	4.7±0.5	5.2±0.8	4.1±0.9
Shakespeare	2.3±0.8	10±3	10±4	10±4	**11±4**

7.3 Scalability of Placement Strategies

Having already established the effectiveness of *Pollen* and analysed the difference between different client placement strategies, we are now concerned with the proposed method's scalability. We begin by comparing with FedScale in a homogeneous setting while increasing the *density* of clients across workers. In this experiment, instead of increasing the number of available GPUs we increase the number of clients per round while keeping the total number of clients at 10 000. This represents a common setting for large-scale

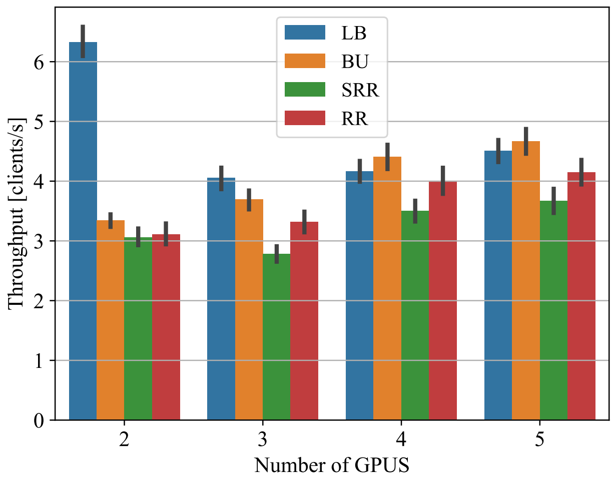

Figure 7: The throughput (the greater, the better) measured in clients per second of Round-Robin (RR), Sorted Round Robin (SRR), Batch-uniform (BU), and Learning-based (LB) client placement policies for different multi-node heterogeneous GPU configurations. The Image Classification task has been used.

Table 2: Here the difference in training time between the fastest and slowest workers (the lower the better) measured in seconds for Round-Robin (RR), Sorted Round Robin (SRR), Batch-uniform (BU), and Learning-based (LB) client placement policies using different multi-node heterogeneous GPU configurations. The Image Classification task has been used.

# GPUs in nodes (0, 1)	BU	LB	RR	SRR
(1, 1)	23±3	**10±4**	30±7	26±5
(1, 2)	21±4	**18±4**	28±6	31±6
(1, 3)	**17±4**	18±4	23±6	23±5
(1, 4)	**15±3**	16±4	21±5	21±6

experiments in which the number of clients per round can be very high. Such setting is often explored by theoretical works and it challenge in a simulation context. As Fig. 8 shows, our push-based placement strategies increase in throughput as the number of clients per round increases. However, the throughput of FedScale does not scale as the number of clients per round increases. This is to be expected since as the number of clients per round increases, *Pollen* does fewer and fewer total communication steps. On the other hand, FedScale does the same number of communication steps. We can also note that the exact placement strategy does not matter, even as the number of clients per round is scaled up, supporting our previous section results.

In the case of heterogeneous GPUs, our results shown in Table 3 indicates that the learning-based solution keeps its advantage over all others even as the number of clients increases. This is driven by the learning-based policy of properly distributing works across GPUs, even when many clients are involved in a round. The other placement strategies have a highly inconsistent ordering as they operate under the false assumption that workers should be treated equally. It is also worth noting that all the improvements in throughput we obtain are based on minimising the wait for the slowest worker at the end of a round. As such, while all placement strategies get an initial boost from increasing client density, once sufficient clients are available each round to keep all workers filled for most of the round, throughput stops increasing.

We now turn our attention to Table 4, which shows the difference between the fastest and slowest worker; we can observe the cause of this trend. When not accounting for hardware heterogeneity, the other policies reflect the speed difference between the GPUs the workers are placed on. It is clear that the learning-based solution minimises this gap relative to the other placement strategies.

Implications: The benefits of client placement for heterogeneous GPU configurations persist as the number of clients processed in a round grows. Importantly, these benefits to be highly consistent after a sufficient number of clients per round is reached.

8 LIMITATIONS

While the results for the push-based client placement system we have presented are compelling and consistent, it does present several limitations, which we list below:

- For homogeneous settings, the improvements are brought by the push-based design, as client placement decisions do not generally matter when using random sampling.
- For heterogeneous settings, a very low number of clients relative to the number of workers does not allow for sufficient placement decisions. Thus, it is difficult for the learning-based method to balance load across workers.
- For vast numbers of clients per round, all placement methods approach their maximum throughput for the task and cease providing further improvements.
- The learning-based policy may not provide the theoretically optimal placement for two reasons. First, even when given a perfect prediction of training time for all clients by an oracle, the distribution of these clients over workers is still non-trivial. Second, the error in estimating each client may cause suboptimal placement decisions regardless of the number of available clients or placement strategy. As we observe from Table 4, the gap between the fastest and slowest worker is always proportional to the number of clients in a fixed manner. This indicates a consistent estimation error.
- The partial aggregation algorithm required to minimise communication is incompatible with some federated learning aggregation algorithms by default. For example, we do not currently support the adaptive optimisation proposed by Reddi et al. [24]

Table 3: The throughput (the greater, the better) for the Image Classification task using *Pollen*'s RR, SRR, BU, and LB client placement strategies with the most heterogeneous multi-node hardware configurations. The number of clients per round has been changed while keeping the total number of trained clients constant.

Num. Clients per round	BU	LB	RR	SRR
100	3.3±0.4	**6±1**	3.1±0.6	3.1±0.5
200	6.7±0.9	**8±1**	7±1	6±1
400	7.1±0.5	**7.9±0.7**	6.9±0.8	7.6±0.5
625	7.1±0.4	**8.4±0.7**	7.2±0.6	6.9±0.4
1000	7.0±0.4	**8.3±0.9**	7.2±0.5	7.4±0.3

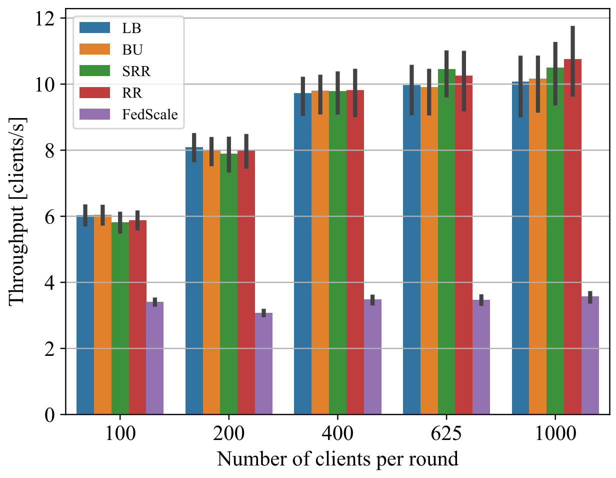

Figure 8: Throughput of the simulation for different placement strategies compared against the fastest framework in performing the Image Classification task. The total number of trained clients has been kept constant to 10000, while the number of clients per round has been increased. The hardware setting used was homogeneous with 4 Nvidia A40.

Table 4: Here the difference in training time between the fastest and slowest workers (the lower the better) for Round-Robin (RR), Sorted Round Robin (SRR), Batch-uniform (BU), and Learning-based (LB) client placement policies using the most heterogeneous multi-node hardware configurations. The number of clients per round has been changed while keeping the total number of trained clients constant. The Image Classification task has been used.

NUM. CLIENTS PER ROUND	BU	LB	RR	SRR
100	23±3	**10±4**	30±7	26±5
200	24±3	**19±4**	30±6	27±5
400	48±3	**41±5**	56±6	43±3
625	77±4	**62±10**	83±8	78±5
1000	125±7	**106±18**	134±11	118±6

9 CONCLUSION

In this work we have shown that the current *pull-based* approaches for client placement available in Federated Learning frameworks are incapable of exploiting both client and hardware heterogeneity. Based on this fact, we have proposed a resource-aware client placement algorithm which minimises communication costs between the server and workers responsible with training the clients. An extensive experimental evaluation has shown our proposed changes to bring improvements of 50% to 400%. This significant improvement allows for the quick development of algorithms with downstream applications in training fleets of millions of edge-devices. Since our modifications have a relatively small footprint, we recommend that all active FL frameworks adopt a similar design for their simulation engines.

REFERENCES

[1] Ravikumar Balakrishnan, Tian Li, Tianyi Zhou, Nageen Himayat, Virginia Smith, and Jeff A. Bilmes. 2022. Diverse Client Selection for Federated Learning via Submodular Maximization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=nwKXyFvaUm

[2] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. 2020. Flower: A Friendly Federated Learning Research Framework. *CoRR* abs/2007.14390 (2020). arXiv:2007.14390 https://arxiv.org/abs/2007.14390

[3] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*, Ameet Talwalkar, Virginia Smith, and Matei Zaharia (Eds.). mlsys.org. https://proceedings.mlsys.org/book/271.pdf

[4] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. LEAF: A Benchmark for Federated Settings. *CoRR* abs/1812.01097 (2018). arXiv:1812.01097 http://arxiv.org/abs/1812.01097

[5] Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. 2021. On Large-Cohort Training for Federated Learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 20461–20475. https://proceedings.neurips.cc/paper/2021/hash/ab9ebd57177b5106ad7879f0896685d4-Abstract.html

[6] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. 2020. Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies. *CoRR* abs/2010.01243 (2020). arXiv:2010.01243 https://arxiv.org/abs/2010.01243

[7] Dimitrios Dimitriadis, Mirian Hipolito Garcia, Daniel Madrigal, Andre Manoel, and Robert Sim. 2022. FLUTE: A Scalable, Extensible Framework for High-Performance Federated Learning Simulations. https://www.microsoft.com/en-us/research/publication/flute-a-scalable-extensible-framework-for-high-performance-federated-learning-simulations/

[8] Wei Gao, Qinghao Hu, Zhisheng Ye, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. 2022. Deep Learning Workload Scheduling in GPU Datacenters: Taxonomy, Challenges and Vision. *CoRR* abs/2205.11913 (2022). https://doi.org/10.48550/arXiv.2205.11913 arXiv:2205.11913

[9] Google. 2019. Tensorflow Federated. https://www.tensorflow.org/federated

[10] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1510.00149

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90

[12] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2021. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.* 14, 1-2 (2021), 1–210. https://doi.org/10.1561/2200000083

[13] Armand K. Koupai, Mohammud Junaid Bocus, Raúl Santos-Rodríguez, Robert J. Piechocki, and Ryan McConville. 2022. Self-Supervised Multimodal Fusion Transformer for Passive Activity Recognition. *CoRR* abs/2209.03765 (2022). https://doi.org/10.48550/arXiv.2209.03765 arXiv:2209.03765

[14] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. 2020. The Open Images Dataset V4. *Int. J. Comput. Vis.* 128, 7 (2020), 1956–1981. https://doi.org/10.1007/s11263-020-01316-z

[15] HyeokHyen Kwon, Catherine Tong, Harish Haresamudram, Yan Gao, Gregory D. Abowd, Nicholas D. Lane, and Thomas Plötz. 2020. IMUTube: Automatic Extraction of Virtual on-body Accelerometry from Video for Human Activity Recognition. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 3 (2020), 87:1–87:29. https://doi.org/10.1145/3411841

[16] Fan Lai, Yinwei Dai, Sanjay Sri Vallabh Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2022. FedScale: Benchmarking Model and System Performance of Federated Learning at Scale. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 11814–11827. https://proceedings.mlr.press/v162/lai22a.html

[17] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proc. VLDB Endow.* 13, 12 (2020), 3005–3018. https://doi.org/10.14778/3415478.3415530

[18] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* 37, 3 (2020), 50–60. https://doi.org/10.1109/MSP.2020.2975749

[19] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and song han. 2022. On-Device Training Under 256KB Memory. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=zGvRdBW06F5

[20] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIV (Lecture Notes in Computer Science, Vol. 11218)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 122–138. https://doi.org/10.1007/978-3-030-01264-9_8

[21] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Xiaojin (Jerry) Zhu (Eds.). PMLR, 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a.html

[22] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, 561–577. https://www.usenix.org/conference/osdi18/presentation/nishihara

[23] Takayuki Nishio and Ryo Yonetani. 2018. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. *CoRR* abs/1804.08333 (2018). arXiv:1804.08333 http://arxiv.org/abs/1804.08333

[24] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. 2021. Adaptive Federated Optimization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=LkFG3lB13U5

[25] Konstantin Sozinov, Vladimir Vlassov, and Sarunas Girdzijauskas. 2018. Human Activity Recognition Using Federated Learning. In *IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, ISPA/IUCC/BDCloud/SocialCom/SustainCom 2018, Melbourne, Australia, December 11-13, 2018*, Jinjun Chen and Laurence T. Yang (Eds.). IEEE, 1103–1111. https://doi.org/10.1109/BDCloud.2018.00164

[26] Catherine Tong, Shyam A. Tailor, and Nicholas D. Lane. 2020. Are Accelerometers for Activity Recognition a Dead-end?. In *HotMobile '20: The 21st International Workshop on Mobile Computing Systems and Applications, Austin, TX, USA, March 3-4, 2020*, Padmanabhan Pillai and Qin Lv (Eds.). ACM, 39–44. https://doi.org/10.1145/3376897.3377867

[27] Ewen Wang, Ajay Kannan, Yuefeng Liang, Boyi Chen, and Mosharaf Chowdhury. 2023. FLINT: A Platform for Federated Learning Integration. *CoRR* abs/2302.12862 (2023). https://doi.org/10.48550/arXiv.2302.12862 arXiv:2302.12862

[28] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR* abs/1804.03209 (2018). arXiv:1804.03209 http://arxiv.org/abs/1804.03209

[29] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao

[30] Han Zhao, Weihao Cui, Quan Chen, Jingwen Leng, Kai Yu, Deze Zeng, Chao Li, and Minyi Guo. 2020. CODA: Improving Resource Utilization by Slimming and Co-locating DNN and CPU Jobs. In *40th*

IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020. IEEE, 853–863. https://doi.org/10.1109/ICDCS47774.2020.00069

References

[1] Ahmed M. Abdelmoniem and Marco Canini. Towards mitigating device heterogeneity in federated learning via adaptive model quantization. In Eiko Yoneki and Paul Patras, editors, *EuroMLSys@EuroSys 2021, Proceedings of the 1st Workshop on Machine Learning and Systemsg Virtual Event, Edinburgh, Scotland, UK, 26 April, 2021*, pages 96–103. ACM, 2021. doi: 10.1145/3437984.3458839. URL `https://doi.org/10.1145/3437984.3458839`.

[2] Ahmed M. Abdelmoniem, Chen-Yu Ho, Pantelis Papageorgiou, and Marco Canini. A comprehensive empirical study of heterogeneity in federated learning. *IEEE Internet of Things Journal*, pages 1–1, 2023. doi: 10.1109/JIOT.2023.3250275.

[3] William Bernardoni, Robert Cardona, Jacob Cleveland, Justin M. Curry, Robert Green, Brian Heller, Alan Hylton, Tung Lam, and Robert Kassouf-Short. Algebraic and geometric models for space networking. *CoRR*, abs/2304.01150, 2023. doi: 10.48550/arXiv.2304.01150. URL `https://doi.org/10.48550/arXiv.2304.01150`.

[4] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. Flower: A friendly federated learning research framework. *CoRR*, abs/2007.14390, 2020. URL `https://arxiv.org/abs/2007.14390`.

[5] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2022.

[6] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In Ameet Talwalkar, Virginia Smith, and Matei Zaharia, editors, *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*. mlsys.org, 2019. URL `https://proceedings.mlsys.org/book/271.pdf`.

[7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html`.

[8] Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. On large-cohort training for federated learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 20461–20475, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/ab9ebd57177b5106ad7879f0896685d4-Abstract.html`.

[9] Fengwen Chen, Guodong Long, Zonghan Wu, Tianyi Zhou, and Jing Jiang. Personalized federated learning with a graph. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 2575–2582. ijcai.org, 2022. doi: 10.24963/ijcai.2022/357. URL `https://doi.org/10.24963/ijcai.2022/357`.

[10] Daiki Chijiwa, Shin'ya Yamaguchi, Yasutoshi Ida, Kenji Umakoshi, and Tomohiro Inoue. Pruning randomly initialized neural networks with iterative randomization. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 4503–4513, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/23e582ad8087f2c03a5a31c125123f9a-Abstract.html`.

[11] Yongheng Deng, Weining Chen, Ju Ren, Feng Lyu, Yang Liu, Yunxin Liu, and Yaoxue Zhang. Tailorfl: Dual-personalized federated learning under system and data heterogeneity. In Jeremy Gummeson, Sunghoon Ivan Lee, Jie Gao, and Guoliang Xing, editors, *Proceedings of the 20th ACM Conference on Embedded Networked*

Sensor Systems, SenSys 2022, Boston, Massachusetts, November 6-9, 2022, pages 592–606. ACM, 2022. doi: 10.1145/3560905.3568503. URL `https://doi.org/10.1145/3560905.3568503`.

[12] Dimitrios Dimitriadis, Mirian Hipolito Garcia, Daniel Madrigal, Andre Manoel, and Robert Sim. Flute: A scalable, extensible framework for high-performance federated learning simulations, March 2022. URL `https://www.microsoft.com/en-us/research/publication/flute-a-scalable-extensible-framework-for-high-performance-federated-learning-`

[13] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011. doi: 10.5555/1953048.2021068. URL `https://dl.acm.org/doi/10.5555/1953048.2021068`.

[14] Chen Dun, Cameron R. Wolfe, Christopher M. Jermaine, and Anastasios Kyrillidis. Resist: Layer-wise decomposition of resnets for distributed training. In James Cussens and Kun Zhang, editors, *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands*, volume 180 of *Proceedings of Machine Learning Research*, pages 610–620. PMLR, 2022. URL `https://proceedings.mlr.press/v180/dun22a.html`.

[15] Chen Dun, Mirian Hipolito Garcia, Chris Jermaine, Dimitrios Dimitriadis, and Anastasios Kyrillidis. Efficient and light-weight federated learning via asynchronous distributed dropout. In Francisco J. R. Ruiz, Jennifer G. Dy, and Jan-Willem van de Meent, editors, *International Conference on Artificial Intelligence and Statistics, 25-27 April 2023, Palau de Congressos, Valencia, Spain*, volume 206 of *Proceedings of Machine Learning Research*, pages 6630–6660. PMLR, 2023. URL `https://proceedings.mlr.press/v206/dun23a.html`.

[16] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In Gal Elidan, Kristian Kersting, and Alexander Ihler, editors, *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. AUAI Press, 2017. URL `http://auai.org/uai2017/proceedings/papers/173.pdf`.

[17] Allen-Jasmin Farcas, Xiaohan Chen, Zhangyang Wang, and Radu Marculescu. Model elasticity for hardware heterogeneity in federated learning systems. In *Proceedings of the 1st ACM Workshop on Data Privacy and Federated Learning Technologies for*

Mobile Edge Network, FedEdge '22, page 19–24, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450395212. doi: 10.1145/3556557.3557954. URL https://doi.org/10.1145/3556557.3557954.

[18] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=6Tm1mposlrM.

[19] Yan Gao, Javier Fernández-Marqués, Titouan Parcollet, Abhinav Mehrotra, and Nicholas D. Lane. Federated self-supervised speech representations: Are we there yet? In Hanseok Ko and John H. L. Hansen, editors, *Interspeech 2022, 23rd Annual Conference of the International Speech Communication Association, Incheon, Korea, 18-22 September 2022*, pages 3809–3813. ISCA, 2022. doi: 10.21437/Interspeech.2022-10644. URL https://doi.org/10.21437/Interspeech.2022-10644.

[20] Zhipeng Gao, Yingwen Duan, Yang Yang, Lanlan Rui, and Chen Zhao. Fedhf: A high fairness federated learning algorithm based on deconfliction in heterogeneous networks. In Javier Troya, Brahim Medjahed, Mario Piattini, Lina Yao, Pablo Fernández, and Antonio Ruiz-Cortés, editors, *Service-Oriented Computing - 20th International Conference, ICSOC 2022, Seville, Spain, November 29 - December 2, 2022, Proceedings*, volume 13740 of *Lecture Notes in Computer Science*, pages 553–566. Springer, 2022. doi: 10.1007/978-3-031-20984-0_39. URL https://doi.org/10.1007/978-3-031-20984-0_39.

[21] Yan Gaol, Javier Fernández-Marqués, Titouan Parcollet, Pedro P. B. de Gusmao, and Nicholas D. Lane. Match to win: Analysing sequences lengths for efficient self-supervised learning in speech and audio. In *IEEE Spoken Language Technology Workshop, SLT 2022, Doha, Qatar, January 9-12, 2023*, pages 115–122. IEEE, 2022. doi: 10.1109/SLT54892.2023.10023410. URL https://doi.org/10.1109/SLT54892.2023.10023410.

[22] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. The non-iid data quagmire of decentralized machine learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 4387–4398. PMLR, 2020. URL http://proceedings.mlr.press/v119/hsieh20a.html.

[23] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *CoRR*, abs/1909.06335, 2019. URL http://arxiv.org/abs/1909.06335.

[24] Zeou Hu, Kiarash Shaloudegi, Guojun Zhang, and Yaoliang Yu. Fedmgda+: Federated learning meets multi-objective optimization. *CoRR*, abs/2006.11489, 2020. URL `https://arxiv.org/abs/2006.11489`.

[25] Tiansheng Huang, Li Shen, Yan Sun, Weiwei Lin, and Dacheng Tao. Fusion of global and local knowledge for personalized federated learning. *Trans. Mach. Learn. Res.*, 2023, 2023. URL `https://openreview.net/forum?id=QtrjqVIZna`.

[26] Wei Huang, Tianrui Li, Dexian Wang, Shengdong Du, Junbo Zhang, and Tianqiang Huang. Fairness and accuracy in horizontal federated learning. *Inf. Sci.*, 589:170–185, 2022. doi: 10.1016/j.ins.2021.12.102. URL `https://doi.org/10.1016/j.ins.2021.12.102`.

[27] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, Kaikai Wang, Anthony Shoumikhin, Jesik Min, and Mani Malek. PAPAYA: practical, private, and scalable federated learning. In Diana Marculescu, Yuejie Chi, and Carole-Jean Wu, editors, *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022. URL `https://proceedings.mlsys.org/paper/2022/hash/f340f1b1f65b6df5b5e3f94d95b11daf-Abstract.html`.

[28] Berivan Isik, Francesco Pase, Deniz Gündüz, Tsachy Weissman, and Michele Zorzi. Sparse random networks for communication-efficient federated learning. *CoRR*, abs/2209.15328, 2022. doi: 10.48550/arXiv.2209.15328. URL `https://doi.org/10.48550/arXiv.2209.15328`.

[29] Xiaopeng Jiang and Cristian Borcea. Complement sparsification: Low-overhead model pruning for federated learning. *CoRR*, abs/2303.06237, 2023. doi: 10.48550/arXiv.2303.06237. URL `https://doi.org/10.48550/arXiv.2303.06237`.

[30] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=SJgIPJBFvH`.

[31] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo,

Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Found. Trends Mach. Learn.*, 14(1-2):1–210, 2021. doi: 10.1561/2200000083. URL `https://doi.org/10.1561/2200000083`.

[32] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5132–5143. PMLR, 2020. URL `http://proceedings.mlr.press/v119/karimireddy20a.html`.

[33] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=H1oyRlYgg`.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[35] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016. URL `http://arxiv.org/abs/1610.05492`.

[36] Fan Lai, Yinwei Dai, Sanjay Sri Vallabh Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 11814–11827. PMLR, 2022. URL `https://proceedings.mlr.press/v162/lai22a.html`.

[37] Harlin Lee, Andrea L. Bertozzi, Jelena Kovacevic, and Yuejie Chi. Privacy-preserving federated multi-task linear regression: A one-shot linear mixing approach inspired

by graph regularization. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*, pages 5947–5951. IEEE, 2022. doi: 10.1109/ICASSP43922.2022.9746007. URL `https://doi.org/10.1109/ICASSP43922.2022.9746007`.

[38] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *CoRR*, abs/2008.03371, 2020. URL `https://arxiv.org/abs/2008.03371`.

[39] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In Jorge Sá Silva, Fernando Boavida, André Rodrigues, Andrew Markham, and Rong Zheng, editors, *SenSys '21: The 19th ACM Conference on Embedded Networked Sensor Systems, Coimbra, Portugal, November 15 - 17, 2021*, pages 42–55. ACM, 2021. doi: 10.1145/3485730.3485929. URL `https://doi.org/10.1145/3485730.3485929`.

[40] Boning Li, Ananthram Swami, and Santiago Segarra. Power allocation for wireless federated learning using graph neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*, pages 5243–5247. IEEE, 2022. doi: 10.1109/ICASSP43922.2022.9747764. URL `https://doi.org/10.1109/ICASSP43922.2022.9747764`.

[41] Dongdong Li. Eb-fedavg: Personalized and training efficient federated learning with early-bird tickets. In Henry Han and Erich Baker, editors, *The Recent Advances in Transdisciplinary Data Science*, pages 213–226, Cham, 2022. Springer Nature Switzerland. ISBN 978-3-031-23387-6.

[42] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, 2020. doi: 10.14778/3415478.3415530. URL `http://www.vldb.org/pvldb/vol13/p3005-li.pdf`.

[43] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.*, 37(3):50–60, 2020. doi: 10.1109/MSP.2020.2975749. URL `https://doi.org/10.1109/MSP.2020.2975749`.

[44] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In Inderjit S.

Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020. URL `https://proceedings.mlsys.org/book/316.pdf`.

[45] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=ByexElSYDr`.

[46] Tian Li, Ahmad Beirami, Maziar Sanjabi, and Virginia Smith. Tilted empirical risk minimization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=K5YasWXZT3O`.

[47] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 6357–6368. PMLR, 2021. URL `http://proceedings.mlr.press/v139/li21h.html`.

[48] Fangshuo Liao and Anastasios Kyrillidis. On the convergence of shallow neural network training with randomly masked neurons. *Trans. Mach. Learn. Res.*, 2022, 2022. URL `https://openreview.net/forum?id=e7mYYMSyZH`.

[49] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Commun. Surv. Tutorials*, 22 (3):2031–2063, 2020. doi: 10.1109/COMST.2020.2986024. URL `https://doi.org/10.1109/COMST.2020.2986024`.

[50] Ji Liu, Stephen J. Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *J. Mach. Learn. Res.*, 16:285–322, 2015. doi: 10.5555/2789272.2789282. URL `https://dl.acm.org/doi/10.5555/2789272.2789282`.

[51] Rui Liu and Han Yu. Federated graph neural networks: Overview, techniques and challenges. *CoRR*, abs/2202.07256, 2022. URL `https://arxiv.org/abs/2202.07256`.

[52] Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Liò, Franco Scarselli, and Andrea Passerini. Graph neural networks for temporal

graphs: State of the art, open challenges, and opportunities. *CoRR*, abs/2302.01018, 2023. doi: 10.48550/arXiv.2302.01018. URL https://doi.org/10.48550/arXiv.2302.01018.

[53] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *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*, pages 1273–1282. PMLR, 2017. URL http://proceedings.mlr.press/v54/mcmahan17a.html.

[54] Chuizheng Meng, Sirisha Rambhatla, and Yan Liu. Cross-node federated graph neural network for spatio-temporal data modeling. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 1202–1211. ACM, 2021. doi: 10.1145/3447548.3467371. URL https://doi.org/10.1145/3447548.3467371.

[55] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4615–4625. PMLR, 2019. URL http://proceedings.mlr.press/v97/mohri19a.html.

[56] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 561–577. USENIX Association, 2018. URL https://www.usenix.org/conference/osdi18/presentation/nishihara.

[57] Luca Moschella, Valentino Maiorca, Marco Fumero, Antonio Norelli, Francesco Locatello, and Emanuele Rodolà. Relative representations enable zero-shot latent space communication. *CoRR*, abs/2209.15430, 2022. doi: 10.48550/arXiv.2209.15430. URL https://doi.org/10.48550/arXiv.2209.15430.

[58] Hamid Mozaffari, Virat Shejwalkar, and Amir Houmansadr. Frl: Federated rank learning, 2022.

[59] Vaikkunth Mugunthan, Eric Lin, Vignesh Gokul, Christian Lau, Lalana Kagal, and Steven D. Pieper. Fedltn: Federated learning for sparse and personalized lottery ticket networks. In Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XII*, volume 13672 of *Lecture Notes in Computer Science*, pages 69–85. Springer, 2022. doi: 10.1007/978-3-031-19775-8_5. URL `https://doi.org/10.1007/978-3-031-19775-8_5`.

[60] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event*, volume 151 of *Proceedings of Machine Learning Research*, pages 3581–3607. PMLR, 2022. URL `https://proceedings.mlr.press/v151/nguyen22b.html`.

[61] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 16784–16804. PMLR, 2022. URL `https://proceedings.mlr.press/v162/nichol22a.html`.

[62] Nobody. Nothing. *None*, 0(0):0, 1970. doi: 00.0000/0000000000. URL `https://example.com`.

[63] Antonio Norelli, Marco Fumero, Valentino Maiorca, Luca Moschella, Emanuele Rodolà, and Francesco Locatello. ASIF: coupled data turns unimodal models to multimodal without training. *CoRR*, abs/2210.01738, 2022. doi: 10.48550/arXiv.2210.01738. URL `https://doi.org/10.48550/arXiv.2210.01738`.

[64] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/arXiv.2303.08774. URL `https://doi.org/10.48550/arXiv.2303.08774`.

[65] Xinchi Qiu, Titouan Parcollet, Javier Fernández-Marqués, Pedro P. B. de Gusmao, Yan Gao, Daniel J. Beutel, Taner Topal, Akhil Mathur, and Nicholas D. Lane. A first look into the carbon footprint of federated learning. *J. Mach. Learn. Res.*, 24:129:1–129:23, 2023. URL `http://jmlr.org/papers/v24/21-0445.html`.

[66] Zhe Qu, Xingyu Li, Rui Duan, Yao Liu, Bo Tang, and Zhuo Lu. Generalized federated learning via sharpness aware minimization. In Kamalika Chaudhuri, Stefanie Jegelka,

Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 18250–18280. PMLR, 2022. URL `https://proceedings.mlr.press/v162/qu22a.html`.

[67] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *https://openai.com/research*, 2018.

[68] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[69] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021. URL `http://proceedings.mlr.press/v139/radford21a.html`.

[70] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR, 2021. URL `http://proceedings.mlr.press/v139/ramesh21a.html`.

[71] Benjamin Recht, Christopher Ré, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 693–701, 2011. URL `https://proceedings.neurips.cc/paper/2011/hash/218a0aefd1d1a4be65601cc6ddc1520e-Abstract.html`.

[72] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. *CoRR*, abs/2003.00295, 2020. URL `https://arxiv.org/abs/2003.00295`.

[73] Yasar Abbas Ur Rehman, Yan Gao, Jiajun Shen, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Federated self-supervised learning for video understanding. In Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXXI*, volume 13691 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2022. doi: 10.1007/978-3-031-19821-2_29. URL `https://doi.org/10.1007/978-3-031-19821-2_29`.

[74] Yifan Shi, Kang Wei, Li Shen, Jun Li, Xueqian Wang, Bo Yuan, and Song Guo. Efficient federated learning with enhanced privacy via lottery ticket pruning in edge computing. *CoRR*, abs/2305.01387, 2023. doi: 10.48550/arXiv.2305.01387. URL `https://doi.org/10.48550/arXiv.2305.01387`.

[75] Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. Detailed comparison of communication efficiency of split learning and federated learning. *CoRR*, abs/1909.09145, 2019. URL `http://arxiv.org/abs/1909.09145`.

[76] Yan Sun, Li Shen, Shixiang Chen, Liang Ding, and Dacheng Tao. Dynamic regularized sharpness aware minimization in federated learning: Approaching global consistency and smooth landscape. *CoRR*, abs/2305.11584, 2023. doi: 10.48550/arXiv.2305.11584. URL `https://doi.org/10.48550/arXiv.2305.11584`.

[77] Zhenheng Tang, Shaohuai Shi, Bo Li, and Xiaowen Chu. Gossipfl: A decentralized federated learning framework with sparsified and adaptive communication. *IEEE Trans. Parallel Distributed Syst.*, 34(3):909–922, 2023. doi: 10.1109/TPDS.2022.3230938. URL `https://doi.org/10.1109/TPDS.2022.3230938`.

[78] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, and Seyit Ahmet Çamtepe. Advancements of federated learning towards privacy preservation: from federated learning to split learning. *CoRR*, abs/2011.14818, 2020. URL `https://arxiv.org/abs/2011.14818`.

[79] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 8485–8493. AAAI Press, 2022. URL `https://ojs.aaai.org/index.php/AAAI/article/view/20825`.

[80] Yuanyishu Tian, Yao Wan, Lingjuan Lyu, Dezhong Yao, Hai Jin, and Lichao Sun. Fedbert: When federated learning meets pre-training. *ACM Trans. Intell. Syst.*

Technol., 13(4):66:1–66:26, 2022. doi: 10.1145/3510033. URL https://doi.org/10.1145/3510033.

[81] Masayoshi Tsutsui and Shinya Takamaeda-Yamazaki. Spins-fl: Communication-efficient federated subnetwork learning. In *20th IEEE Consumer Communications & Networking Conference, CCNC 2023, Las Vegas, NV, USA, January 8-11, 2023*, pages 605–610. IEEE, 2023. doi: 10.1109/CCNC51644.2023.10060698. URL https://doi.org/10.1109/CCNC51644.2023.10060698.

[82] Zheng Wang, Xiaoliang Fan, Jianzhong Qi, Chenglu Wen, Cheng Wang, and Rongshan Yu. Federated learning with fair averaging. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1615–1623. ijcai.org, 2021. doi: 10.24963/ijcai.2021/223. URL https://doi.org/10.24963/ijcai.2021/223.

[83] Cameron R. Wolfe, Jingkang Yang, Arindam Chowdhury, Chen Dun, Artun Bayer, Santiago Segarra, and Anastasios Kyrillidis. GIST: distributed training for large-scale graph convolutional networks. *CoRR*, abs/2102.10424, 2021. URL https://arxiv.org/abs/2102.10424.

[84] Zheshun Wu, Xiaoping Wu, and Yunliang Long. Multi-level federated graph learning and self-attention based personalized wi-fi indoor fingerprint localization. *IEEE Commun. Lett.*, 26(8):1794–1798, 2022. doi: 10.1109/LCOMM.2022.3159504. URL https://doi.org/10.1109/LCOMM.2022.3159504.

[85] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386. URL https://doi.org/10.1109/TNNLS.2020.2978386.

[86] Pengwei Xing, Songtao Lu, Lingfei Wu, and Han Yu. Big-fed: Bilevel optimization enhanced graph-aided federated learning. *IEEE Transactions on Big Data*, pages 1–12, 2022. doi: 10.1109/TBDATA.2022.3191439.

[87] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. Asynchronous federated learning on heterogeneous devices: A survey. *CoRR*, abs/2109.04269, 2021. URL https://arxiv.org/abs/2109.04269.

[88] Ashkan Yousefpour, Shen Guo, Ashish Shenoy, Sayan Ghosh, Pierre Stock, Kiwan Maeng, Schalk-Willem Krüger, Michael G. Rabbat, Carole-Jean Wu, and Ilya Mironov. Green federated learning. *CoRR*, abs/2303.14604, 2023. doi: 10.48550/arXiv.2303.14604. URL https://doi.org/10.48550/arXiv.2303.14604.

[89] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/3fe78a8acf5fda99de95303940a2420c-Abstract.html`.

[90] Binhang Yuan, Cameron R. Wolfe, Chen Dun, Yuxin Tang, Anastasios Kyrillidis, and Chris Jermaine. Distributed learning of fully connected neural networks using independent subnet training. *Proc. VLDB Endow.*, 15(8):1581–1590, 2022. URL `https://www.vldb.org/pvldb/vol15/p1581-wolfe.pdf`.

[91] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P. Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 1171–1180. ACM, 2017. doi: 10.1145/3038912.3052660. URL `https://doi.org/10.1145/3038912.3052660`.

[92] Manzil Zaheer, Sashank J. Reddi, Devendra Singh Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9815–9825, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/90365351ccc7437a1309dc64e4db32a3-Abstract.html`.

[93] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *CoRR*, abs/1806.00582, 2018. URL `http://arxiv.org/abs/1806.00582`.

[94] Hanhan Zhou, Tian Lan, Guru Prasadh Venkataramani, and Wenbo Ding. Federated learning with online adaptive heterogeneous local models. In *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*, 2022. URL `https://openreview.net/forum?id=p3EhUXVMeyn`.