FedScale: Benchmarking Model and System Performance of Federated Learning at Scale

Fan Lai [1] Yinwei Dai [1] Sanjay S. Singapuram [1] Jiachen Liu [1] Xiangfeng Zhu [1 2] Harsha V. Madhyastha [1]
Mosharaf Chowdhury [1]

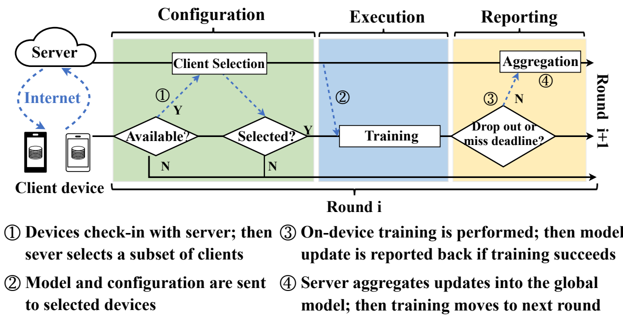
Abstract

We present FedScale, a federated learning (FL) benchmarking suite with realistic datasets and a scalable runtime to enable reproducible FL research. FedScale datasets encompass a wide range of critical FL tasks, ranging from image classification and object detection to language modeling and speech recognition. Each dataset comes with a unified evaluation protocol using real-world data splits and evaluation metrics. To reproduce realistic FL behavior, FedScale contains a scalable and extensible runtime. It provides high-level APIs to implement FL algorithms, deploy them at scale across diverse hardware and software backends, and evaluate them at scale, all with minimal developer efforts. We combine the two to perform systematic benchmarking experiments and highlight potential opportunities for heterogeneity-aware co-optimizations in FL. FedScale is open-source and actively maintained by contributors from different institutions at http://fedscale.ai. We welcome feedback and contributions from the community.

1. Introduction

Federated learning (FL) is an emerging machine learning (ML) setting where a logically centralized coordinator orchestrates many distributed clients (e.g., smartphones or laptops) to collaboratively train or evaluate a model (Bonawitz et al., 2019; Kairouz et al., 2021b) (Figure 1). It enables model training and evaluation on end-user data, while circumventing high cost and privacy risks in gathering the raw data from clients, with applications across diverse ML tasks.

In the presence of heterogeneous execution speeds of client



① Devices check-in with server; then sever selects a subset of clients
② Model and configuration are sent to selected devices
③ On-device training is performed; then model update is reported back if training succeeds
④ Server aggregates updates into the global model; then training moves to next round

Figure 1. Standard FL protocol.

devices as well as non-IID data distributions, existing efforts have focused on optimizing different aspects of FL: (1) *System efficiency*: reducing computation load (e.g., using smaller models (Sandler et al., 2018)) or communication traffic (e.g., local SGD (McMahan et al., 2017)) to achieve shorter round duration; (2) *Statistical efficiency*: designing data heterogeneity-aware algorithms (e.g., client clustering (Ghosh et al., 2020b)) to obtain better training accuracy with fewer training rounds; (3) *Privacy and security*: developing reliable strategies (e.g., differentially private training (Kairouz et al., 2021a)) to make FL more privacy-preserving and robust to potential attacks.

A comprehensive benchmark to evaluate an FL solution must investigate its behavior under the practical FL setting with (1) *data heterogeneity* and (2) *device heterogeneity* under (3) *heterogeneous connectivity* and (4) *availability* conditions at (5) *multiple scales* on a (6) *broad variety of ML tasks*. While the first two aspects are oft-mentioned in the literature (Li et al., 2020), realistic network connectivity and the availability of client devices can affect both heterogeneities (e.g., distribution drift (Eichner et al., 2019)), impairing model convergence. Similarly, evaluation at a large scale can expose an algorithm's robustness, as practical FL deployment often runs across thousands of concurrent participants out of millions of clients (Yang et al., 2018). Overlooking any one aspect can mislead FL evaluation (§2).

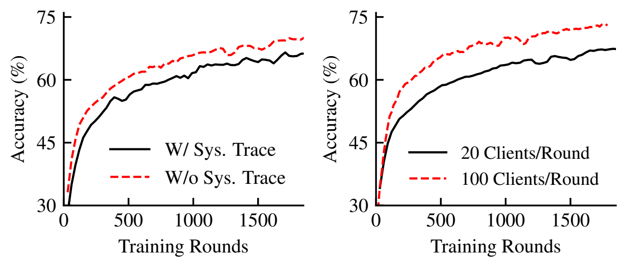
Unfortunately, existing FL benchmarks often fall short across multiple dimensions (Table 1). First, they are limited in the versatility of data for various real-world FL appli-

[1]Department of Computer Science, University of Michigan [2]Department of Computer Science, University of Washington. Correspondence to: Fan Lai <fanlai@umich.edu>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

Features	LEAF	TFF	FedML	Flower	**FedScale**
Heter. Client Dataset	◯	✗	◯	◯	✔
Heter. System Speed	✗	✗	◯	◯	✔
Client Availability	✗	✗	✗	✗	✔
Scalable Platform	✗	✔	◯	✔	✔
Real FL Runtime	✗	✗	✗	✗	✔
Flexible APIs	✗	✔	✔	✔	✔

Table 1. Comparing FedScale with existing FL benchmarks and libraries. ◯ implies limited support.



(a) Impact of system trace. (b) Impact of scale.

Figure 2. Existing benchmarks can be misleading. We train ShuffleNet on OpenImage classification (Detailed setup in Section 5).

cations. Indeed, even though they may have quite a few datasets and FL training tasks (e.g., LEAF (Caldas et al., 2019)), their datasets often contain synthetically generated partitions derived from conventional datasets (e.g., CIFAR) and do not represent realistic characteristics. This is because these benchmarks are mostly borrowed from traditional ML benchmarks (e.g., MLPerf (Mattson et al., 2020)) or designed for simulated FL environments like TensorFlow Federated (TFF) (tff) or PySyft (pys). Second, existing benchmarks often overlook system speed, connectivity, and availability of the clients (e.g., FedML (He et al., 2020) and Flower (Beutel et al., 2021)). This discourages FL efforts from considering system efficiency and leads to overly optimistic statistical performance (§2). Third, their datasets are primarily small-scale, because their experimental environments are unable to emulate large-scale FL deployments. While real FL often involves thousands of participants in each training round (Kairouz et al., 2021b; Yang et al., 2018), most existing benchmarking platforms can merely support the training of tens of participants per round; Finally, most of them lack user-friendly APIs for automated integration, resulting in great engineering efforts for benchmarking at scale. We attached the detailed comparison of existing benchmarks with FedScale in Appendix C.

Contributions We introduce an FL benchmark, FedScale, to enable comprehensive and standardized FL evaluations:

- To the best of our knowledge, we incorporate the most comprehensive FL datasets for evaluating different aspects of real FL deployments. FedScale currently has 20 realistic FL datasets spanning across small, medium, and large scales for a wide variety of task categories, such as image classification, object detection, word prediction, speech recognition, and reinforcement learning. To account for practical client behaviors, we include real-world measurements of mobile devices, and associate each client with his computation and communication speeds, as well as the availability status over time.
- We build an automated evaluation platform, FedScale Runtime, to simplify and standardize the FL evaluation in a more realistic setting. FedScale Runtime provides a mobile backend to enable on-device FL evaluation, and a cluster backend to benchmark various practical FL metrics (e.g., real client round duration) on GPUs/CPUs using real FL statistical and system dataset. The cluster backend can perform the training of thousands of clients in each round on a few GPUs efficiently, and allows easy deployment of new plugins with flexible APIs.
- We perform systematic experiments to show how FedScale facilitates today's FL benchmarking, and highlight the pressing need of co-optimizing system and statistical efficiency, especially in tackling system stragglers, biased model accuracy, and device energy trade-offs.

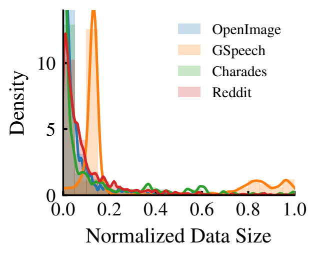
2. Background

Existing efforts for various goals of practical FL To tackle heterogeneous client data, FedProx (Li et al., 2020), FedYogi (Reddi et al., 2020) and Scaffold (Karimireddy et al., 2020) introduce adaptive client/server optimizations that use control variates to account for the 'drift' in model updates. Instead of training a single global model, some efforts train a mixture of models (Shi et al., 2021; Fallah et al., 2020), cluster clients over training (Ghosh et al., 2020a), or enforce guided client selection (Lai et al., 2021); To tackle the scarce and heterogeneous device resource, FedAvg (McMahan et al., 2017) reduces communication cost by performing multiple local SGD steps, while some works compress the model update by filtering out or quantizing unimportant parameters (Rothchild et al., 2020; Karimireddy et al., 2019); After realizing the privacy risk in FL (Geiping et al., 2020; Wang et al., 2020), DP-SGD (Geyer et al., 2017) enhances the privacy by employing differential privacy, and DP-FTRL (Kairouz et al., 2021a) applies the tree aggregation to add noise to the sum of mini-batch gradients. These FL efforts often navigate accuracy-computation-privacy trade-offs. As such, a realistic FL setting is crucial for comprehensive evaluations.

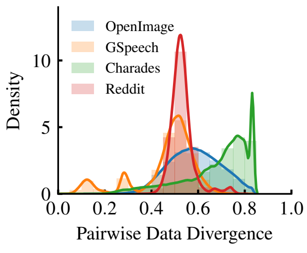
Existing FL benchmarks can be misleading Existing benchmarks often lack realistic client statistical and system behavior datasets, and/or fail to reproduce large-scale FL

Category	Name	Data Type	#Clients	#Instances	Example Task
CV	*OpenImage*	Image	13,771	1.3M	Classification, Object detection
	Google Landmark	Image	43,484	3.6M	Classification
	Charades	Video	266	10K	Action recognition
	VLOG	Video	4,900	9.6K	Classification, Object detection
	Waymo Motion	Video	496,358	32.5M	Motion prediction
NLP	*Europarl*	Text	27,835	1.2M	Text translation
	Reddit	Text	1,660,820	351M	Word prediction
	LibriTTS	Text	2,456	37K	Text to speech
	Google Speech	Audio	2,618	105K	Speech recognition
	Common Voice	Audio	12,976	1.1M	Speech recognition
Misc ML	*Taobao*	Text	182,806	20.9M	Recommendation
	Puffer Streaming	Text	121,551	15.4M	Sequence prediction
	Fox Go	Text	150,333	4.9M	Reinforcement learning

Table 2. Statistics of *partial* FedScale datasets (the full list of data and its partition are available in Appendix B). FedScale has 20 real-world federated datasets; each dataset is partitioned by its real client-data mapping, and we have removed sensitive information in these datasets.



(a) Data size.



(b) Data distribution.

Figure 3. Non-IID client data.

deployments. Unfortunately, these limitations imply that they are not only insufficient for benchmarking diverse FL optimizations, but can even mislead performance evaluations: (1) As shown in Figure 2(a), the statistical performance becomes worse when encountering practical client behaviors (e.g., client training failures and availability dynamics), which indicates existing benchmarks that do not have systems traces can produce overly optimistic statistical performance; (2) FL training with hundreds of participants each round performs better than that with tens of participants (Figure 2(b)). As such, existing benchmark platforms can under-report FL optimizations as they cannot support the practical FL scale with a large number of participants.

3. FedScale Dataset: Realistic FL Workloads

We next introduce how we build realistic datasets in FedScale to fulfill the desired properties of FL datasets, such as the client statistical dataset and system traces.

3.1. Client Statistical Dataset

FedScale currently has 20 realistic FL datasets (Table 2) across diverse practical FL scenarios and disciplines. For example, Puffer dataset (Yan et al., 2020) is from FL video streaming deployed to edge users over the Internet. The raw data of FedScale datasets are collected from different sources and stored in various formats. We clean up the raw data, partition them into new FL datasets, streamline new datasets into consistent formats, and categorize them into different FL use cases. Moreover, FedScale provides standardized APIs, a Python package, for the user to easily leverage these datasets (e.g., using different distributions of the same data or new datasets) in other frameworks.

Realistic data and partitions We target realistic datasets with client information, and partition the raw dataset using the unique client identification. For example, OpenImage is a vision dataset collected by Flickr, wherein different mobile users upload their images to the cloud for public use. We use the *AuthorProfileUrl* attribute of the OpenImage data to map data instances to each client, whereby we extract the realistic distribution of the raw data. Following the practical FL deployments (Yang et al., 2018), we assign the clients of each dataset into the training, validation and testing groups, to get its training, validation and testing set. Here, we pick four real-world datasets – video (Charades), audio (Google Speech), image (OpenImage), and text (Reddit) – to illustrate practical FL characteristics. Each dataset consists of hundreds or up to millions of clients and millions of data points. Figure 3 reports the *Probability Density Function* (PDF) of the data distribution, wherein we see a high statistical deviation (e.g., wide distribution of the density) across clients not only in the quantity of samples (Figure 3(a)) but also in the data distribution (Figure 3(b)).[1] We notice that realistic datasets mostly have unique Non-IID patterns, implying the impracticality of existing artificial FL partitions.

Different scales across diverse task categories To accommodate diverse scenarios in practical FL, FedScale includes small-, medium-, and large-scale datasets across a wide range of tasks, from hundreds to millions of clients. Some datasets can be applied in different tasks, as we en-

[1]We report the pairwise Jensen–Shannon distance of the label distribution between two clients.

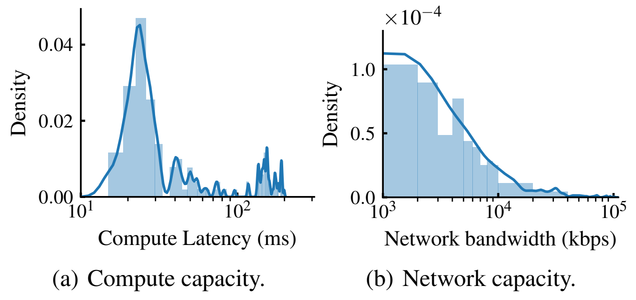


Figure 4. Heterogeneous client system speed.

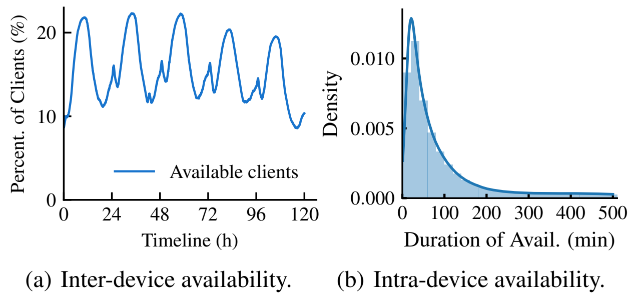


Figure 5. Client availability is dynamic.

rich their use case by deriving different metadata from the same raw data. For example, the raw OpenImage dataset can be used for object detection, and we extract each object therein and generate a new dataset for image classification. Moreover, we provide APIs for the developer to customize their dataset (e.g., enforcing new data distribution or taking a subset of clients for evaluations with a smaller scale).

3.2. Client System Behavior Dataset

Client device system speed is heterogeneous We formulate the system trace of different clients using *AI Benchmark* (Ignatov et al., 2019) and *MobiPerf Measurements* (mob) on mobiles. *AI Benchmark* provides the training and inference speed of diverse models (e.g., MobileNet) across a wide range of device models (e.g., Huawei P40 and Samsung Galaxy S20), while *MobiPerf* has collected the available cloud-to-edge network throughput of over 100k world-wide mobile clients. As specified in real FL deployments (Bonawitz et al., 2019; Yang et al., 2018), we focus on mobile devices that have larger than 2GB RAM and connect with WiFi; Figure 4 reports that their compute and network capacity can exhibit order-of-magnitude difference. As such, how to orchestrate scarce resources and mitigate stragglers are paramount.

Client device availability is dynamic We incorporate a large-scale user behavior dataset spanning across 136k users (Yang et al., 2021) to emulate the behaviors of clients. It includes 180 million trace items of client devices (e.g., battery charge or screen lock) over a week. We follow the real FL setting, which considers the device in charging to be available (Bonawitz et al., 2019) and observe great dynamics in their availability: (i) the number of available clients reports diurnal variation (Figure 5(a)). This confirms the cyclic patterns in the client data, which can deteriorate the statistical performance of FL (Eichner et al., 2019). (ii) the duration of each available slot is not long-lasting (Figure 5(b)). This highlights the need of handling failures (e.g., clients become offline) during training, as the round duration (also a few minutes) is comparable to that of each available slot. This, however, is largely overlooked in the literature.

4. FedScale Runtime: Evaluation Platform

Existing FL evaluation platforms can hardly reproduce the practical FL scale. Worse, they often lack user-friendly APIs, requiring great developer efforts to deploy new plugins. As such, we introduce, FedScale Runtime, an automated and easily-deployable evaluation platform equipped with mobile and cluster backends, to simplify and standardize the FL evaluation under a practical setting.

4.1. FedScale Runtime: Mobile Backend

FedScale Runtime deploys a mobile backend to enable on-device FL evaluation on smartphones. The first principle, in building our mobile backend, is to minimize any engineering effort for the developer (e.g., without reinventing their Python code) to benchmark FL on mobiles. To this end, FedScale mobile backend is built atop the Termux app (ter), an Android terminal that supports Linux environment.

```
from fedscale.core.client import Client

class Mobile_Client(Client):
  def train(self,client_data,model,conf):
    for local_step in range(conf.local_steps):
      optimizer.zero_grad()
      ...
      loss.backward()
      optimizer.step()
    # Results will be sent to cloud aggregator via gRPC
    return gradient_update
```

Figure 6. Training on mobile client.

Figure 6 shows a snippet of code running on FedScale mobile backend. By integrating with Termux, FedScale Runtime allows the developer to run an unmodified version of Python script (e.g., PyTorch) built from source on the mobile device, and the full-operator set (e.g., PyTorch Modules) is available too. This eases the deployment cycle, in that FL models and algorithms that were prototyped on server GPUs/CPUs, can also be deployed using FedScale Runtime. We are currently implementing the Google Remote Procedure Call (gRPC) for distributed mobile devices to interface

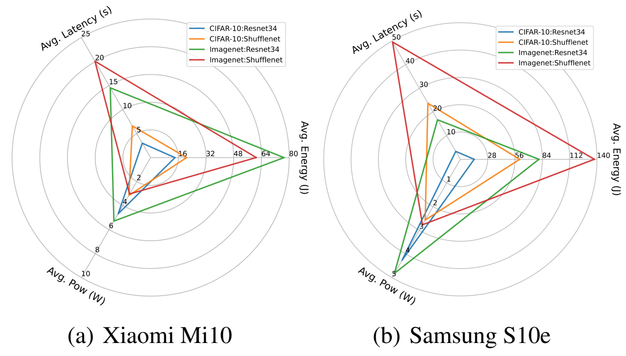


Figure 7. FedScale Runtime can benchmark the mobile runtime of power, energy and latency. We train Resnet34 and Shufflenet on ImageNet and CIFAR-10 on Xiaomi mi10 and Samsung S10e.

with FedScale Runtime cloud server.

Benchmarking Mobile Runtime FedScale mobile backend enables the developer to benchmark realistic FL training/testing performance on mobile phones. For example, Figure 7 reports the performance metrics of training Shufflenet and Resnet34 on one mini-batch (batch size 32), drawn from the Imagenet and CIFAR-10 datasets, on Xiaomi Mi10 and Samsung S10e Android devices. We benchmark the average training time. We notice that Resnet34 runs at higher instantaneous power than Shufflenet on both devices, but it requires less total energy to train since it takes shorter latency. ImageNet takes longer than CIFAR-10 per mini-batch, as the larger training image sizes lead to longer execution. The heterogeneity in computational capacity is evident as the Xiaomi Mi10 device outperforms the Samsung S10e device due to a more capable processor. As such, we believe that FedScale mobile backend can facilitate future on-device FL optimizations (e.g., hardware-aware Neural Architecture Search (He et al., 2018)).

4.2. FedScale Runtime: Simulator

However, benchmarking on mobile devices is often prohibitively expensive. As such, FedScale Runtime provides an automated simulator that performs FL training/testing on GPUs/CPUs, while providing various practical FL metrics by emulating realistic FL behaviors, such as computation/communication cost, latency and wall clock time. To the best of our knowledge, FedScale Runtime is the first platform that enables FL benchmarking with practical FL runtime on GPUs/CPUs.

Overview of FedScale Runtime Simulator FedScale Runtime primarily consists of three components (Figure 8):

- *Aggregator Simulator*: It acts as the aggregator in practical FL, which selects participants, distributes execution profiles (e.g., model weight), and handles result (e.g., model updates) aggregation. In each round, its client manager uses the client behavior trace to monitor whether a client is available; then it selects the specified number of clients to participate that round. Once receiving new events, the event monitor activates the handler (e.g., aggregation handler to perform model aggregation), or the gRPC communicator to send/receive messages. The communicator records the size (cost) of every network traffic, and its runtime latency in FL wall-clock time ($\frac{traffic\_size}{client\_bandwidth}$).
- *Client Simulator*: It works as the FL client. FedScale data loader loads the federated dataset of that client and feeds this data to the compute engine to run real training/testing. The computation latency is determined by ($\#\_processed\_sample \times latency\_per\_sample$), and the communicator handles network traffics and records the communication latency ($\frac{traffic\_size}{client\_bandwidth}$). The device monitor will terminate the simulation of a client if the current FL runtime exceeds his available slot (e.g., client drops out), indicated by the availability trace.
- *Resource Manager*: It orchestrates the available physical resource for evaluation to maximize the resource utilization. For example, when the number of participants/round exceeds the resource capacity (e.g., simulating thousands of clients on a few GPUs), the resource manager queues the overcommitted tasks of clients and schedules new client simulation from this queue once resource becomes available. Note that this queuing will not affect the simulated FL runtime, as this runtime is controlled by global virtual clock, and the event monitor will manage events in the correct FL runtime order.

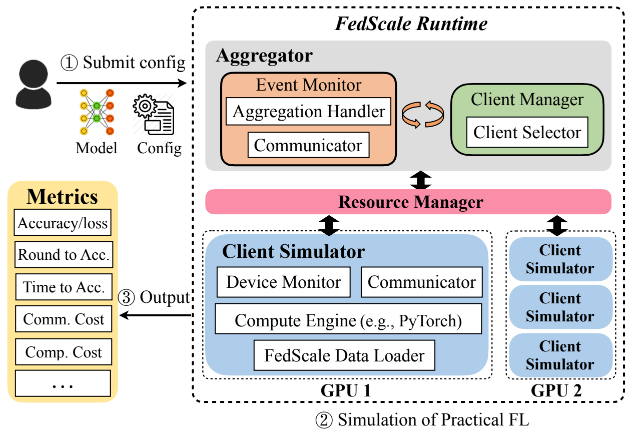


Figure 8. FedScale Runtime enables the developer to benchmark various FL efforts with practical FL data and metrics.

Note that capturing runtime performance (e.g., wall clock time) is rather slow and expensive in practical FL – each mobile device takes several minutes to train a round – but our simulator enables *fast-forward* simulation, as training on CPUs/GPUs takes only a few seconds per round, while

Module	API Name	Example Use Case
Aggregator Simulator	`round_completion_handler`()	Adaptive/secure aggregation
	`client_completion_handler`()	Straggler mitigation
Client Manager	`select_participants`()	Client selection
	`select_model_for_client`()	Adaptive model selection
Client Simulator	`train`()	Local SGD/malicious attack
	`serialize_results`()	Model compression

Table 3. Some example APIs. FedScale provides APIs to deploy new plugins for various designs. We omit input arguments for brevity here.

```
from fedscale.core.client import Client

class Customized_Client(Client):
# Redefine training (e.g., for local
    SGD/gradient compression)

  def train(self,client_data,model,conf):
    # Code of plugin
        ...

    # Results will be serialized, and
        then sent to aggregator
    return training_result
```

Figure 9. Add plugins by inheritance.

providing simulated runtime using realistic traces.

FedScale Runtime enables automated FL simulation FedScale Runtime incorporates realistic FL traces, using the aforementioned trace by default or the developer-specified profile from the mobile backend, to automatically emulate the practical FL workflow: ① *Task submission*: FL developers specify their configurations (e.g., model and dataset), which can be federated training or testing, and the resource manager will initiate the aggregator and client simulator on available resource (GPU, CPU, other accelerators, or even smartphones); ② *FL simulation*: Following the standardized FL lifecycle (Figure 1), in each training round, the aggregator inquires the client manager to select participants, whereby the resource manager distributes the client configuration to the available client simulators. After the completion of each client, the client simulator pushes the model update to the aggregator, which then performs the model aggregation. ③ *Metrics output*: During training, the developer can query the practical evaluation metrics on the fly. Figure 8 lists some popular metrics in FedScale.

FedScale Runtime is easily-deployable and extensible for plugins FedScale Runtime provides flexible APIs, which can accommodate with different execution backends (e.g., PyTorch) by design, for the developer to quickly benchmark new plugins. Table 3 illustrates some example APIs that can facilitate diverse FL efforts, and Figure 9 dictates an example showing how these APIs help to benchmark a new design of local client training with a few lines of code by inheriting the base Client module. Moreover, FedScale Runtime can embrace new realistic (statistical client or system behavior) datasets with the built-in APIs. For example, the developer can import his own dataset of the client availability with the API (`load_client_availability`), and FedScale Runtime will automatically enforce this trace during evaluations. We provide more examples and the comparison with other frameworks, in Appendix D to show the ease of evaluating various today's FL work in FedScale Runtime– a few lines are all we need!

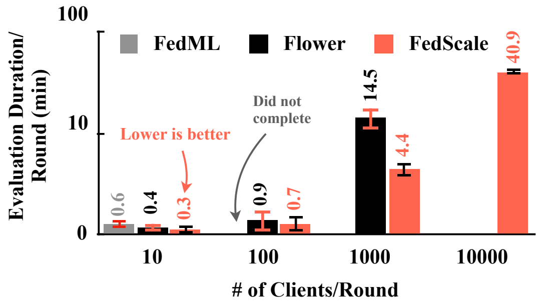


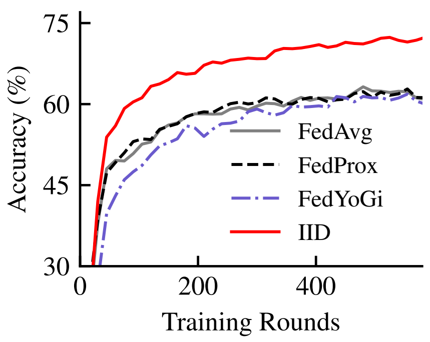
Figure 10. FedScale Runtime can run thousands of clients/round, while FedML and Flower failed to run large-scale clients on 10 GPUs. We have more scalability experiments in Appendix C.

FedScale Runtime is scalable and efficient FedScale Runtime simulator can perform large-scale simulations (thousands of clients per round) in both standalone (single CPU/GPU) and distributed (multiple machines) settings. This is because: (1) FedScale Runtime can support multiprocessing on GPUs so that multiple client simulators can co-locate on the same GPU; (2) our resource manager monitors the fine-grained resource utilization of machines, queues the overcommitted simulation requests, adaptively dispatches simulation requests of the client across machines to achieve load balance, and then orchestrates the simulation based on the client virtual clock. Instead, state-of-the-art platforms can hardly support the practical FL scale, due to their limited support for distributed simulations (e.g., FedJax (Ro et al., 2021)), and/or the reliance on the traditional ML architecture that trains on a few workers with long-running computation, whereas FedScale Runtime minimizes the overhead (e.g., frequent data serialization) in the fleet training of FL clients. As shown in Figure 10 [2], other than being able to evaluate the practical FL runtime, FedScale Runtime not only runs faster than FedML (He et al., 2020) and Flower (Beutel et al., 2021), but can support large-scale evaluations efficiently.

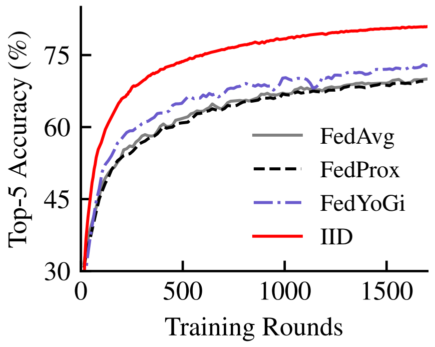
[2]We train ShuffleNet on OpenImage classification task on 10 GPU nodes. Detailed experimental setups in Appendix A.

Task	Dataset	Model	IID	FedAvg	FedProx	FedYogi
Image Classification	FEMNIST	ResNet-18	86.40%	**78.50%**	78.40%	76.30%
	OpenImage	ShuffleNet-V2	81.37%	70.27%	69.54%	**74.04%**
		MobileNet-V2	80.83%	70.09%	70.34%	**75.25%**
Text Classification	Amazon Review	Logistic Regression	66.10%	**65.80%**	65.10%	65.30%
Language Modeling	Reddit	Albert	73.5 perplexity (ppl)	77.3 ppl	**76.6 ppl**	81.6 ppl
Speech Recognition	Google Speech	ResNet-34	72.58%	**63.37%**	63.25%	62.67%

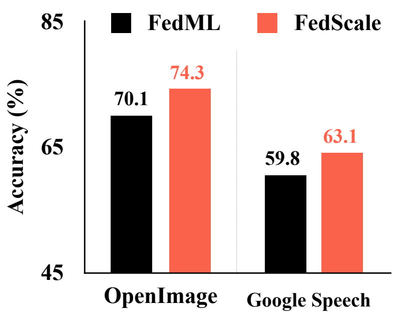
Table 4. Benchmarking of different FL algorithms across realistic FL datasets. We report the mean test accuracy over 5 runs.



(a) Convergence on Google speech.



(b) Convergence on OpenImage.



(c) Final model performance.

Figure 11. FedScale can benchmark the statistical FL performance. (c) shows existing benchmarks can under-report the FedYoGi performance as they cannot support a large number of participants.

5. Experiments

In this section, we show how FedScale can facilitate better benchmarking of FL efforts over its counterparts.

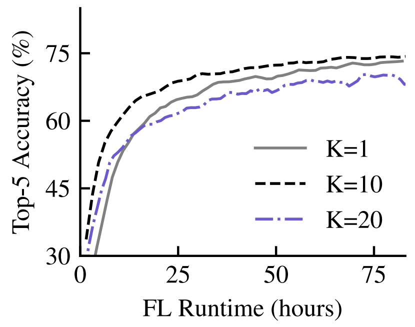
Experimental setup We use 10 NVIDIA Tesla P100 GPUs in our evaluations. Following the real FL deployments (Bonawitz et al., 2019; Yang et al., 2018), the aggregator collects updates from the first N completed participants out of $1.3N$ participants to mitigate system stragglers in each round, and $N = 100$ by default. We experiment with representative FedScale datasets in different scales and tasks (detailed experiment setup in Appendix A).

5.1. How Does FedScale Help FL Benchmarking?

Existing benchmarks are insufficient to evaluate the various metrics needed in today's FL. We note that the performance of existing benchmarks and FedScale are quite close in the same settings if we turn off the optional system traces in FedScale. Because the underlying training and FL protocols in evaluations are the same. However, the limited scalability can mislead the practical FL performance. Next, we show the effectiveness of FedScale in benchmarking different FL aspects over its counterparts.

Benchmarking FL statistical efficiency. FedScale provides various realistic client datasets to benchmark the FL statistical efficiency. Here, we experiment with state-of-the-art optimizations (FedAvg, FedProx and FedYoGi) – each aims to mitigate the data heterogeneity – and the traditional IID data setting. Figure 11 and Table 4 report that: (1) the round-to-accuracy performance and final model accuracy of the non-IID setting is worse than that of the IID setting, which is consistent with existing findings (Kairouz et al., 2021b); (2) different tasks can have different preferences on the optimizations. For example, FedYoGi performs the best on OpenImage, but it is inferior to FedAvg on Google Speech. With much more FL datasets, FedScale enables extensive studies of the sweet spot of different optimizations; and (3) existing benchmarks can under-report the FL performance due to their inability to reproduce the FL setting. Figure 11(c) reports the final model accuracy using FedML and FedScale, where we attempt to reproduce the scale of practical FL with 100 participants per round in both frameworks, but FedML can only support 30 participants because of its suboptimal scalability, which under-reports the FL performance that the algorithm can indeed achieve.

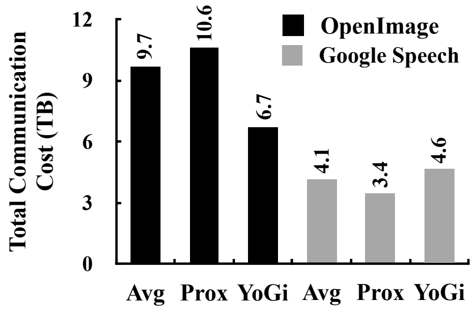
Benchmarking FL system efficiency. Existing system optimizations for FL focus on the practical runtime (e.g., wall-clock time in real FL training) and the FL execution cost. Unfortunately, existing benchmarks can hardly evaluate the FL runtime due to the lack of realistic system traces, but we now show how FedScale can help such benchmark-



(a) FedScale reports realistic FL clock.

Setting	FL runtime (hours)	Evaluation time (hours)
K=1	97	17
K=10	52	9
K=20	130	47

(b) FedScale enables fast-forward eval.



(c) FedScale reports FL communication cost.

Figure 12. FedScale can benchmark realistic FL runtime. (a) and (b) report FedYoGi results on OpenImage with different number of local steps (K); (b) reports the FL runtime to reach convergence.

ing: (1) FedScale Runtime enables fast-forward evaluations of the practical FL wall-clock time with fewer evaluation hours. Taking different number of local steps K in local SGD as an example (McMahan et al., 2017), Figure 12(a) and Table 12(b) illustrate that FedScale can evaluate this impact of K on practical FL runtime in a few hours. This allows the developer to evaluate large-scale system optimizations efficiently; and (2) FedScale Runtime can dictate the FL execution cost by using realistic system traces. For example, Figure 12(c) reports the practical FL communication cost in achieving the performance of Figure 11, while Figure 15 reports the system duration of individual clients. These system metrics can facilitate developers to navigate the accuracy-cost trade-off.

Benchmarking FL privacy and security. FedScale can evaluate the statistical and system efficiency for privacy and security optimizations more realistically. Here, we give an example of benchmarking the DP-SGD (Geyer et al., 2017; Kairouz et al., 2021a), which applies differential privacy to improve the client privacy. We experiment with different privacy targets σ (σ=0 indicates no privacy enhancement) and different number of participants per round N. Figure 13 shows that the scale of participants (e.g., N=30) that today's benchmarks can support can mislead the privacy evaluations too: for σ=0.01, while we notice great performance degradation (12.8%) in the final model accuracy when N=30, this enhancement is viable in practical FL (N=100) with decent accuracy drop (4.6%). Moreover, FedScale is able to benchmark more practical FL metrics, such as wall-clock time, communication cost added in privacy optimizations, and the number of rounds needed to leak the client privacy under realistic individual client data and Non-IID distributions.

As for benchmarking the FL security, we follow the example setting of recent backdoor attacks (Sun et al., 2019; Wang et al., 2020) on the OpenImage, where corrupted clients flip

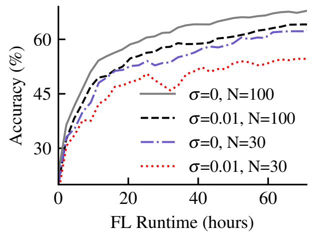


Figure 13. FedScale can benchmark privacy efforts in more realistic FL settings.

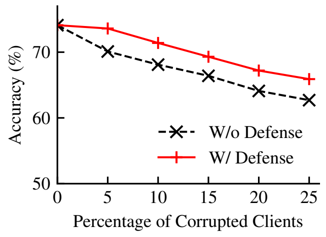


Figure 14. FedScale can benchmark security optimizations with realistic FL data.

their ground-truth labels to poison the training. We benchmarked two settings: one without security enhancement, while the other clips the model updates as (Sun et al., 2019). As shown in Figure 14, while state-of-the-art optimizations report this can mitigate the attacks without hurting the overall performance on their synthesized dataset, we notice a great accuracy drop in more practical FL settings.

5.2. Opportunities for Future FL Optimizations

Next, we show FedScale can shine light on the need for yet unexplored optimizations owing to its realistic FL settings.

Heterogeneity-aware co-optimizations of communication and computation Existing optimizations for the system efficiency often apply the same strategy on all clients (e.g., using the same number of local steps (McMahan et al., 2017) or compression threshold (Rothchild et al., 2020)), while ignoring the heterogeneous client system speed. When we outline the timeline of 5 randomly picked participants in training of the ShuffleNet (Figure 15), we find: (1) system stragglers can greatly slow down the round aggregation in practical FL; and (2) simply optimizing the communication or computation efficiency may not lead to faster rounds, as the last participant can be bottlenecked by the other resource.

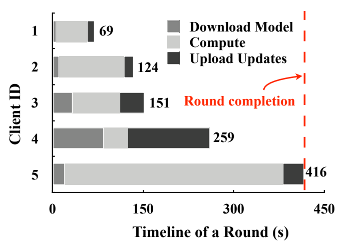


Figure 15. System stragglers slow down practical FL greatly.

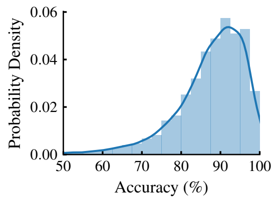


Figure 16. Biased accuracy distributions of the trained model across clients (Shufflenet on OpenImage).

Here, optimizing the communication can greatly benefit *Client 4*, but it achieves marginal improvement on the round duration as *Client 5* is computation-bound. This implies an urgent need of heterogeneity-aware co-optimizations of communication and computation efficiency.

Co-optimizations of statistical and system efficiency Most of today's FL efforts focus on either optimizing the statistical or the system efficiency, whereas we observe a great opportunity to jointly optimize both efficiencies: (1) As the system behavior determines the availability of client data, predictable system performance can benefit statistical efficiency. For example, in alleviating the biased model accuracy (Figure 16), we may prioritize the use of upcoming offline clients to curb the upcoming distribution drift of client data; (2) Statistical optimizations should be aware of the heterogeneous client system speed. For example, instead of applying one-fit-all strategies (e.g., local steps or gradient compression) for all clients, faster workers can trade more system latency for more statistical benefits (e.g., transferring more traffics with less intensive compression).

FL design-decisions considering mobile environment Existing efforts have largely overlooked the interplay of client devices and training speed (e.g., using a large local steps to save communication (McMahan et al., 2017)), however, as shown in Figure 7, running intensive on-device computation for a long time can quickly drain the battery, or even burn the device, leading to the unavailability of clients. Therefore, we believe that a power and temperature-aware training algorithm (e.g., different local steps across clients or device-aware NAS) can be an important open problem.

6. Conclusion

To enable scalable and reproducible FL research, we introduce FedScale, a diverse set of realistic FL datasets in terms of scales, task categories and client system behaviors, along with a more scalable evaluation platform, FedScale Runtime, than the existing. FedScale Runtime performs fast-forward evaluation of the practical FL runtime metrics needed in today's work. More subtly, FedScale Runtime provides ready-to-use realistic datasets and flexible APIs to allow more FL applications, such as benchmarking NAS, model inference, and a broader view of federated computation (e.g., multi-party computation). FedScale is open-source at http://fedscale.ai, and we hereby invite the community to develop and contribute state-of-the-art FL efforts.

Acknowledgments

We would like to thank the anonymous reviewers and SymbioticLab members for their insightful feedback. We also thank FedScale contributors and users from many different academic institutions and industry for their valuable feedbacks to improve FedScale. This work was supported in part by NSF grants CNS-1900665, CNS-1909067, and CNS-2106184.

References

Common Voice Data. https://commonvoice.mozilla.org/en/datasets.

Fox go dataset. https://github.com/featurecat/go-dataset.

iNaturalist 2019. https://sites.google.com/view/fgvc6/competitions/inaturalist-2019.

MobiPerf. https://www.measurementlab.net/tests/mobiperf/.

Google Open Images Dataset. https://storage.googleapis.com/openimages/web/index.html.

PySyft. https://github.com/OpenMined/PySyft.

Reddit Comment Data. https://files.pushshift.io/reddit/comments/.

Stack Overflow Data. https://cloud.google.com/bigquery/public-data/stackoverflow.

Taobao Dataset. https://tianchi.aliyun.com/dataset/dataDetail?dataId=56&lang=en-us.

Termux. https://termux.com/.

TensorFlow Federated. https://www.tensorflow.org/federated.

Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Parcollet, T., de Gusmao, P. P. B., and Lane, N. D. FLOWER: A friendly federated learning framework. *arXiv preprint arXiv:2007.14390*, 2021.

Bonawitz, K., Eichner, H., and et al. Towards federated learning at scale: System design. In *MLSys*, 2019.

Caldas, S., Meher, S., Duddu, K., and et al. Leaf: A benchmark for federated settings. *NeurIPS' Workshop*, 2019.

Chai, D., Wang, L., Chen, K., and Yang, Q. FedEval: A benchmark system with a comprehensive evaluation model for federated learning. In *arxiv.org/abs/2011.09655*, 2020.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. In *arxiv.org/abs/1702.05373*, 2017.

Eichner, H., Koren, T., McMahan, H. B., Srebro, N., and Talwar, K. Semi-cyclic stochastic gradient descent. In *ICML*, 2019.

Ettinger, S., Cheng, S., Caine, B., Liu, C., Zhao, H., Pradhan, S., Chai, Y., Sapp, B., Qi, C., Zhou, Y., Yang, Z., Chouard, A., Sun, P., Ngiam, J., Vasudevan, V., McCauley, A., Shlens, J., and Anguelov, D. Large scale interactive motion forecasting for autonomous driving : The waymo open motion dataset. *CoRR*, abs/2104.10133, 2021.

Fallah, A., Mokhtari, A., and Ozdaglar, A. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.

Fouhey, D. F., Kuo, W., Efros, A. A., and Malik, J. From lifestyle vlogs to everyday interactions. In *CVPR*, 2018.

Geiping, J., Bauermeister, H., Dröge, H., and Moeller, M. Inverting gradients - how easy is it to break privacy in federated learning? In *NeurIPS*, 2020.

Geyer, R. C., Klein, T., and Nabi, M. Differentially private federated learning: A client level perspective. In *NeurIPS*, 2017.

Ghosh, A., Chung, J., Yin, D., and Ramchandran, K. An efficient framework for clustered federated learning. In *NeurIPS*, 2020a.

Ghosh, A., Chung, J., Yin, D., and Ramchandran, K. An efficient framework for clustered federated learning. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020b.

He, C., Li, S., So, J., and Zeng, X. FedML: A research library and benchmark for federated machine learning. In *arxiv.org/abs/2007.13518*, 2020.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018.

Ignatov, A., Timofte, R., Kulik, A., Yang, S., Wang, K., Baum, F., Wu, M., Xu, L., and Gool, L. V. AI benchmark: All about deep learning on smartphones in 2019. *CoRR*, abs/1910.06663, 2019. URL http://arxiv.org/abs/1910.06663.

Kairouz, P., McMahan, B., Song, S., Thakkar, O., Thakurta, A., and Xu, Z. Practical and private (deep) learning without sampling or shuffling. In *arxiv.org/abs/2103.00039*, 2021a.

Kairouz, P., McMahan, H. B., and et al. Advances and open problems in federated learning. In *Foundations and Trends® in Machine Learning*, 2021b.

Karimireddy, S. P., Rebjock, Q., Stich, S. U., and Jaggi, M. Error feedback fixes signsgd and other gradient compression schemes. In *arXiv preprint arXiv:1901.09847*, 2019.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. SCAFFOLD: Stochastic controlled averaging for federated learning. In *ICML*, 2020.

Koehn, P. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, 2005.

Lai, F., Zhu, X., Madhyastha, H. V., and Chowdhury, M. Oort: Efficient federated learning via guided participant selection. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021.

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

Mao, H., Netravali, R., and Alizadeh, M. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pp. 197–210, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346535. doi: 10.1145/3098822.3098843. URL https://doi.org/10.1145/3098822.3098843.

Mattson, P., Cheng, C., Coleman, C., and et al. Mlperf training benchmark. In *MLSys*, 2020.

McAuley, J., Targett, C., Shi, Q., and van den Hengel, A. Image-based recommendations on styles and substitutes. In *SIGIR*, 2015.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

Reddi, S., Charles, Z., and et al. Adaptive federated optimization. In *arxiv.org/abs/2003.00295*, 2020.

Reddy, S., Chen, D., and Manning, C. D. Coqa: A conversational question answering challenge. *arXiv preprint arXiv:1808.07042*, 2019.

Ro, J. H., Suresh, A. T., and Wu, K. Fedjax: Federated learning simulation with jax, 2021.

Rothchild, D., Panda, A., Ullah, E., Ivkin, N., Stoica, I., Braverman, V., Gonzalez, J., and Arora, R. Fetchsgd: Communication-efficient federated learning with sketching. In *ICML*, 2020.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Schler, J., Koppel, M., Argamon, S., and Pennebaker, J. Effects of age and gender on blogging. In *Proceedings of AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs*, 2006.

Shi, N., Lai, F., Kontar, R. A., and Chowdhury, M. Fed-ensemble: Improving generalization through model ensembling in federated learning, 2021.

Sigurdsson, G. A., Varol, G., Wang, X., Farhadi, A., Laptev, I., and Gupta, A. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *ECCV*, 2016.

Sun, Z., Kairouz, P., Suresh, A. T., and McMahan, H. B. Can you really backdoor federated learning. In *arxiv.org/abs/1911.07963*, 2019.

Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., yong Sohn, J., Lee, K., and Papailiopoulos, D. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS*, 2020.

Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. In *arxiv.org/abs/1804.03209*, 2018.

Weyand, T., Araujo, A., Cao, B., and Sim, J. Google landmarks dataset v2 a large-scale benchmark for instance-level recognition and retrieval. In *arxiv.org/abs/2004.01804*, 2020.

Yan, F. Y., Ayers, H., and et al. Learning in situ: a randomized experiment in video streaming. In *NSDI*, 2020.

Yang, C., Wang, Q., and et al. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In *WWW*, 2021.

Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., and Beaufays, F. Applied federated learning: Improving Google keyboard query suggestions. In *arxiv.org/abs/1812.02903*, 2018.

Zen, H., Dang, V., Clark, R., Zhang, Y., Weiss, R. J., Jia, Y., Chen, Z., and Wu, Y. Libritts: A corpus derived from librispeech for text-to-speech. *arXiv preprint arXiv:1904.02882*, 2019.

A. Experiment Setup

Scalability Evaluations We evaluate the scalability of FedScale Runtime, FedML (GitHub *commit@2ee0517*) and Flower (v0.17.0 atop Ray v1.9.2) using a cluster with 10 GPU nodes. Each GPU node has a P100 GPU with 12GB GPU memory and 192GB RAM. We train the ShuffleNet-V2 model on the OpenImage dataset. We set the minibatch size of each participant to 32, and the number of local steps K to 20, which takes around 2800MB GPU memory for each model training. As such, we allow each GPU node to run 4 processes in benchmarking these three frameworks.

Evaluation Setup Applications and models used in our evaluations are widely used on mobile devices. We set the minibatch size of each participant to 32, and the number of local steps K to 20. We cherry-pick the hyper-parameters with grid search, ending up with an initial learning rate 0.04 for CV tasks and 5e-5 for NLP tasks. The learning rate decays by 0.98 every 10 training rounds. These settings are consistent with the literature. More details about the input dataset are available in Appendix B.

B. Introduction of FedScale Datasets

FedScale currently has 20 realistic federated datasets across a wide range of scales and task categories (Table 5). Here, we provide the description of some representative datasets.

Google Speech Commands. A speech recognition dataset (Warden, 2018) with over ten thousand clips of one-second-long duration. Each clip contains one of the 35 common words (e.g., digits zero to nine, "Yes", "No", "Up", "Down") spoken by thousands of different people.

OpenImage. OpenImage (ope) is a vision dataset collected from Flickr, an image and video hosting service. It contains a total of 16M bounding boxes for 600 object classes (e.g., Microwave oven). We clean up the dataset according to the provided indices of clients. In our evaluation, the size of each image is 256×256.

Reddit and StackOverflow. Reddit (red) (StackOverflow (sta)) consists of comments from the Reddit (StackOverflow) website. It has been widely used for language modeling tasks, and we consider each user as a client. In this dataset, we restrict to the 30k most frequently used words, and represent each sentence as a sequence of indices corresponding to these 30k frequently used words.

VLOG. VLOG (Fouhey et al., 2018) is a video dataset collected from YouTube. It contains more than 10k Lifestyle Vlogs, videos that people purportedly record to show their lives, from more than 4k actors. This dataset is aimed at understanding everyday human interaction and contains labels for scene classification, hand-state prediction, and hand detection tasks.

LibriTTS. LibriTTS (Zen et al., 2019) is a large-scale text-to-speech dataset. It is derived from audiobooks that are part of the LibriVox project. There are 585 hours of read English speech from 2456 speakers at a 24kHz sampling rate.

Taobao. Taobao Dataset (tao) is a dataset of click rate prediction about display Ad, which is displayed on the website of Taobao. It is composed of 1,140,000 users ad display/click logs for 8 days, which are randomly sampled from the website of Taobao. We partitioned it using its real client-data mapping.

Waymo Motion. Waymo Motion (Ettinger et al., 2021) is composed of 103,354 segments each containing 20 seconds of object tracks at 10Hz and map data for the area covered by the segment. These segments are further broken into 9 second scenarios (8 seconds of future data and 1 second of history) with 5 second overlap, and we consider each scenario as a client.

Puffer Streaming. Puffer is a Stanford University research study about using machine learning (e.g., reinforcement learning (Mao et al., 2017)) to improve video-streaming algorithms: the kind of algorithms used by services such as YouTube, Netflix, and Twitch. Puffer dataset (Yan et al., 2020) consists of 15.4M sequences of network throughput on edge clients over time.

We will make FedScale open-source on the Github. So the code and dataset can be downloaded from the repository. For each dataset, we will provide detailed descriptions (`README.md`) of the source, organization, format and use case under the repository. In the future, these datasets will be hosted on our server, or be migrated to the stable storage of AWS. For the evaluation platform FedScale Runtime, we will provide the configuration and job submission guidelines as well.

C. Comparison with Existing FL Benchmarks

In this section, we compare FedScale with existing FL benchmarks in more details.

Data Heterogeneity Existing benchmarks for FL are mostly limited in the variety of realistic datasets for real-world FL applications. Even they have various datasets (e.g., LEAF (Caldas et al., 2019)) and FedEval (Chai et al., 2020)), their datasets are mostly synthetically derived from conventional datasets (e.g., CIFAR) and limited to quite a

Category	Name	Data Type	#Clients	#Instances	Example Task
CV	*iNature* (ina)	Image	2,295	193K	Classification
	FEMNIST (Cohen et al., 2017)	Image	3,400	640K	Classification
	OpenImage (ope)	Image	13,771	1.3M	Classification, Object detection
	Google Landmark (Weyand et al., 2020)	Image	43,484	3.6M	Classification
	Charades (Sigurdsson et al., 2016)	Video	266	10K	Action recognition
	VLOG (Fouhey et al., 2018)	Video	4,900	9.6K	Classification, Object detection
	Waymo Motion (Ettinger et al., 2021)	Video	496,358	32.5M	Motion prediction
NLP	*Europarl* (Koehn, 2005)	Text	27,835	1.2M	Text translation
	Blog Corpus (Schler et al., 2006)	Text	19,320	137M	Word prediction
	Stackoverflow (sta)	Text	342,477	135M	Word prediction, Classification
	Reddit (red)	Text	1,660,820	351M	Word prediction
	Amazon Review (McAuley et al., 2015)	Text	1,822,925	166M	Classification, Word prediction
	CoQA (Reddy et al., 2019)	Text	7,189	114K	Question Answering
	LibriTTS (Zen et al., 2019)	Text	2,456	37K	Text to speech
	Google Speech (Warden, 2018)	Audio	2,618	105K	Speech recognition
	Common Voice (com)	Audio	12,976	1.1M	Speech recognition
Misc ML	*Taxi Trajectory*	Text	442	1.7M	Sequence prediction
	Taobao (tao)	Text	182,806	20.9M	Recommendation
	Puffer Streaming (Yan et al., 2020)	Text	121,551	15.4M	Sequence prediction
	Fox Go (go-)	Text	150,333	4.9M	Reinforcement learning

Table 5. Statistics of FedScale datasets. FedScale has 20 realistic client datasets, which are from the real-world collection, and we partitioned each dataset using its real client-data mapping.

few FL tasks. These statistical client datasets can not represent realistic characteristics and are inefficient to benchmark various real FL applications. Instead, FedScale provides 20 comprehensive realistic datasets for a wide variety of tasks and across small, medium, and large scales, and these datasets can also be used in data analysis to motivate more FL designs.

System Heterogeneity The practical FL statistical performance also depends on the system heterogeneity (e.g., client system speed and availability of the client), which has inspired lots of optimizations for FL system efficiency. However, existing FL benchmarks have largely overlooked the system behaviors of FL clients, which can produce misleading evaluations, and discourage the benchmarking of system efforts. To emulate the heterogeneous system behaviors in practical FL, FedScale incorporates real-world traces of mobile devices, and associates each client with his system speeds, as well as the availability. Moreover, it is non-trivial to emulate these behaviors at scale, so we develop FedScale Runtime, which is more efficient than the existing ones.

Scalability Existing frameworks, perhaps due to the heavy burden of building complicated system support, largely rely on the traditional ML architectures (e.g., the primitive parameter-server architecture of Pytorch). These architectures are primarily designed for the traditional large-batch training on a number of workers, and each worker often trains a single batch at a time. However, this is ill-suited to the simulation of thousands of clients concurrently: (1) they lack tailored system implementations to orchestrate the synchronization and resource scheduling, for which they can easily run into synchronization/memory issues and crash down; (2) their resource can be under-utilized, as FL evaluations often use a much smaller batch size than that in the traditional architecture.

Tackling all these inefficiencies requires domain-specific system designs. Specifically, we first built an advanced resource scheduler: It monitors the fine-grained resource utilization of machines, queues the overcommitted simulation requests, adaptively dispatches simulation requests of the client across machines to achieve load balance, and then orchestrates the simulation based on the client virtual clock. Moreover, given a much smaller batch size in FL, we maximize the resource utilization by overlapping the communication and computation phrases of different client simulations. The former and the latter make FedScale more scalable across machines and on single machines, respectively.

Empirically, we have run the 20-GPU set up on different datasets and models in Figure 10 and Table 6, and are aware of at least one group who ran FedScale Runtime with more than 60 GPUs (Lai et al., 2021).

	Eval. Duration/Round vs. # of Clients/Round			
	10	100	1K	10K
FedScale	**0.03 min**	**0.16 min**	**1.14 min**	**10.9 min**
FedML	0.58 min	4.4 min	fail to run	fail to run
Flower	0.05 min	0.23 min	2.4 min	fail to run

Table 6. FedScale is more scalable and faster. Image classification on iNature dataset using MobileNet-V2 on 20-GPU setting.

Modularity As shown in Table 1, some existing frameworks (e.g., LEAF and FedEval) do not provide user-friendly modularity, which requires great engineering efforts to benchmark different components, and we recognize that FedML and Flower provide the API modularity in this table too.

On the other hand, FedScale Runtime's modularity for easy deployments and broader use cases is not limited to APIs (Figure 8): (1) FedScale Runtime Data Loader: it simplifies and expands the use of realistic datasets. e.g., developers can load and analyze the realistic FL data to motivate new algorithm designs, or imports new datasets/customize data distributions in FedScale evaluations; (2) Client simulator: it emulates the system behaviors of FL clients, and developers can customize their system traces in evaluating the FL system efficiency too; (3) Resource Manager: it hides the system complexity in training large-scale participants simultaneously for the deployment.

D. Examples of New Plugins

```
from fedscale.core.client_manager import ClientManager
import Oort

class Customized_ClientManager(ClientManager):
  def __init__(self, *args):
    super().__init__(*args)
    self.oort_selector =
        Oort.create_training_selector(*args)

  # Replace default client selection algorithm w/ Oort
  def select_participants(self, numOfClients, cur_time,
      feedbacks):
    # Feed Oort w/ execution feedbacks from last
        training round
    oort_selector.update_client_info(feedbacks)
    selected_clients =
        oort_selector.select_participants(numOfClients,
        cur_time)

    return selected_clients
```

Figure 17. Evaluate new client selection algorithm (Lai et al., 2021).

In this section, we demonstrate several examples to show the ease of integrating today's FL efforts for realistic evaluations in FedScale.

At its core, FedScale Runtime provides flexible APIs on

```
from fedscale.core.client import Client

class Customized_Client(Client):
# Customize the training on each client
  def train(self,client_data,model,conf):
    # Get the training result from
    # the default training component
    training_result = super().train(
        client_data, model, conf)

    # Implementation of compression
    compressed_result = compress_impl(
            training_result)
    return compressed_result
```

Figure 18. Evaluate model compression (Rothchild et al., 2020).

```
from fedscale.core.client import Client

class Customized_Client(Client):
# Customize the training on each client
  def train(self,client_data,model,conf):
    # Get the training result from
    # the default training component
    training_result = super().train(
        client_data, model, conf)

    # Clip updates and add noise
    secure_result = secure_impl(
            training_result)
    return secure_result
```

Figure 19. Evaluate security enhancement (Sun et al., 2019).

each module so that the developer can access and customize methods of the base class. Table 3 illustrates some example APIs that can facilitate diverse FL efforts. Note that FedScale Runtime will automatically integrate new plugins into evaluations, and then produces practical FL metrics. Figure 17 demonstrates that we can easily evaluate new client selection algorithms, Oort (Lai et al., 2021), by modifying a few lines of the `clientManager` module. Similarly, Figure 18 and Figure 19 show that we can extend the basic `Client` module to apply new gradient compression (Rothchild et al., 2020) and enhancement for malicious attack (Sun et al., 2019), respectively.

Comparison with other work Figure 20 shows the same evaluation of gradient compression (Rothchild et al., 2020) as that in Figure 18 using flower (Beutel et al., 2021), which requires much more human efforts than Figure 18. The gray components in figure 20 requires implementation. Flower falls short in providing APIs for passing metadata between client and server, for example client id, which makes server and running workers client-agnostic. To customized anything for clients during FL training, developers have to go through the source code and override many components to share client meta data between server and workers.

```
import argparse
import flwr as fl

def get_config_fn():
  def fit_config(rnd: int):
    # Implementation of randomly selection
    client_ids = random_selection()
    config = {"ids": client_ids}
    return config

  return fit_config
# Customized Strategy
strategy = CustomizedStrategy(
  on_fit_config_fn=get_config_fn(),
)

fl.server.start_server(
  config={"num_rounds":args.round},
  strategy=strategy)
```

```
import argparse
import flwr as fl

class Customized_Client():
  def fit(self, config, net):
    # Customization of client data
    trainloader = select_dataset(
      config["ids"][args.partition])
    train(net, trainloader)
    compressed_result = self.get_parameters()
    # Implementation of compression
    compressed_result = compress_impl(
        training_result)
    return compressed_result

fl.client.start_numpy_client(
  args.address,
  client=CustomizedClient())
```

Figure 20. Evaluate model compression with flower (Beutel et al., 2021). The developer needs to implement the functions in grey by his own. Note that each function can take tens of lines of code.