Heterogeneity-Aware Federated Learning

Chengxu Yang
Peking University
yangchengxu@pku.edu.cn

Qipeng Wang
Beihang University
wangqipeng0924@gmail.com

Mengwei Xu
Peking University
mwx@pku.edu.cn

Shangguang Wang
Beijing University of Posts and Telecommunications
sgwang@bupt.edu.cn

Kaigui Bian
Peking University
bkg@pku.edu.cn

Xuanzhe Liu
Peking University
liuxuanzhe@pku.edu.cn

ABSTRACT

Federated learning (FL) is an emerging distributed machine learning paradigm that stands out with its inherent privacy-preserving advantages. Heterogeneity is one of the core challenges in FL, which resides in the diverse user behaviors and hardware capacity across devices who participate in the training. Heterogeneity inherently exerts a huge influence on the FL training process, e.g., causing device unavailability. However, Existing FL literature usually ignores the impacts of heterogeneity. To fill in the knowledge gap, we build FLASH, the first heterogeneity-aware FL platform. Based on FLASH and a large-scale user trace from 136k real-world users, we demonstrate the usefulness of FLASH in anatomizing the impacts of heterogeneity in FL by exploring three previously unaddressed research questions: whether and how can heterogeneity affect FL performance; how to configure a heterogeneity-aware FL system; and what are heterogeneity's impacts on existing FL optimizations. It shows that heterogeneity causes nontrivial performance degradation in FL from various aspects, and even invalidates some typical FL optimizations.

1 INTRODUCTION

In the past decade, we have witnessed the explosion of machine learning (ML) and especially deep learning (DL) applications [55] that are built upon "big data". Recently, there has been increasing attention paid to data privacy [53]. For example, there have been various regulations on privacy protections, such as GDPR [54], CCPA [51], and China Internet Security Law [52]. As a result, existing ML and DL applications have to face a grand challenge, i.e., data, especially private data, cannot be arbitrarily collected and used for ML and DL model training.

To this end, Federated Learning (FL) is proposed, where a set of client devices collaboratively learn a model without sharing private data [8, 32]. FL takes the advantages in preserving user privacy, as it does not require a centralized data collection. Both academia and industry have made a lot of efforts in pushing forward FL research. For example, Tensorflow [1] and PyTorch [41], dominantly popular ML frameworks, are taking steps to support FL. Meanwhile, various FL platforms [9, 40, 47, 50, 56, 56] have been proposed to facilitate FL development.

In practice, deploying FL is rather complicated [5]. One key challenge is the uncertainties in terms of performance introduced by the so-called **heterogeneity**, which can hinder the deployment of FL in two aspects.

• **Behavior heterogeneity**: users' behavior over devices can be quite diverse from one another, and can affect the learning process that relies on the device status, e.g., idle, charging, or connected to an unmetered network such as Wi-Fi [5]. As a result, participants are unreliable and can drop out at any time, leading to uneven participation over the training group [26].

• **Device heterogeneity**: the participating devices can vary a lot in terms of hardware specifications like computation, battery, and communication capabilities. Intuitively, low-end devices and unreliable network conditions may compromise the performance of FL [29, 30].

To our best knowledge, existing FL platforms [9, 40, 47, 50, 56] do not incorporate the heterogeneity into their design. For example, *TensorFlow Federated* [50] enables developers to evaluate novel federated algorithms on integrated Non-I.I.D. datasets, while assuming that all the clients are available at any time. As a result, the existing optimization of heterogeneity leaves a knowledge gap on whether or to what degree will heterogeneity influence FL in the real world. To this end, we are motivated to further explore the influence of device and behavior heterogeneity in FL.

To comprehensively examine the influence of heterogeneity, we propose FLASH, the first platform that incorporates and simulates the heterogeneity in federated learning. For behavior heterogeneity, we deploy a commodity input method app and collect a large-scale real-world user behavior trace that covers 136k users and spans one week. For device heterogeneity, we first perform the on-device training on mobile devices with different hardware capacities and then use the results of AI-Benchmark [19] as an intermediary to profile

the devices in our trace (more than 1,000 types of device models).

Base on the collected data, we conduct extensive experiments over FLASH (>5,700 GPU-hrs) to characterize heterogeneity's impacts on FL. We use three synthetic benchmarks [14, 36, 44], which are widely used in FL literature [9, 23, 27, 28, 32], and the user input corpora from the input method app. We summarize our research questions (RQs) and findings as follows:

RQ 1: Whether and how can heterogeneity affect FL performance? We find that heterogeneity makes a non-trivial impact on the performance of FL tasks: on average 4% (up to 9.2%) accuracy drop and 1.74× (up to 2.32×) convergence slowdown (§3.1). To break down, behavior heterogeneity introduces severe participation bias and accuracy bias (§3.2), which compromises the model accuracy accordingly. Device heterogeneity causes the participated clients to fail in the model training, which slows down the model convergence and leads to local resources consumption (§3.3). The results indicate that a heterogeneity-aware platform like FLASH is necessary for developers to precisely understand how their model shall perform in real-world settings. It also urges researchers to develop more efficient FL protocols to hedge the performance degradation due to heterogeneity.

RQ 2: How to configure a heterogeneity-aware FL system? Some meaningful configurations are introduced heterogeneity [57], including reporting deadline, goal client count, etc. (§2.6). However, these configurations are seldom studied in previous efforts. We find that a tight or distant deadline can slow down the model convergence, and an optimal deadline exists at a middle point that can be obtained by monitoring the client failure rate (§4.1). In our experiments, selecting $100 - 300$ clients for each round can produce the best FL performance [9, 21, 27, 28]. This setting contradicts to some reported efforts [9, 21, 27, 28], where much fewer (usually tens of) clients are selected and the reported accuracy is 7.4% lower than ours, i.e., 0.75 *vs.* 0.81 (§4.2). The results highlight the necessity to properly configure the FL system, as a misconfiguration can significantly degrade the FL performance.

RQ 3: What are heterogeneity's impacts on existing FL optimizations? Various FL optimizations are evaluated, but the impact of heterogeneity is not comprehensively considered [10, 27, 38] or even missing [23, 28, 38, 43]. To examine whether the existing optimizations are still valid given heterogeneity, we explore two typical mechanisms, i.e., gradient compression algorithms [2, 4, 23] and aggregation algorithms [26, 28]. We reproduce a few representative optimizations and find that the novel aggregation algorithms are not quite effective with heterogeneity considered. For example, behavior heterogeneity hinders *q-FedAvg* [28] from addressing the fairness issues in FL (§5.2). We also find that current gradients compression algorithms cannot speed up FL convergence, especially under heterogeneity-aware environment (§5.1). In the worst cases, the convergence time is lengthened by 3.5×. The above findings suggest that existing optimizations should be carefully validated on heterogeneity-aware settings, and new ones should be explored.

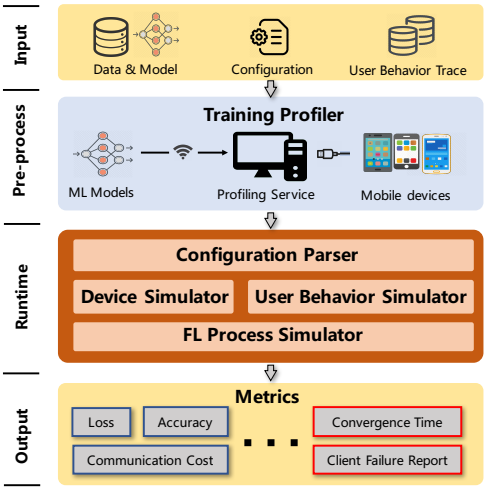

Figure 1: FLASH workflow

We summarize our contributions as follows.

- We propose FLASH, an open platform for simulating heterogeneity-aware FL, which mitigates the gap between FL development and real-world deployment.
- We carry out extensive experiments to demonstrate the impacts of heterogeneity in FL tasks. Our results provide useful and insightful implications for future research in designing more efficient FL systems.
- We have released the large scale user behavior trace and our source code implementation[1] to facilitate future FL research.

2 FLASH DESIGN

2.1 Design Principles

FLASH is an FL simulation platform for developers and researchers, which enforces the following principles.

- **Heterogeneity-aware**. The platform should consider heterogeneity, i.e., it should include behavior heterogeneity and device heterogeneity that are ignored in all existing platforms.
- **Flexible settings**. The platform should suggest the necessary configuration settings to simulate different FL environments that reflect heterogeneity.
- **Comprehensive metrics**. The platform should provide the simulation results from various aspects so that

[1] An anonymous repository: https://github.com/imc20submission/flash

the developers can have a comprehensive and insightful understanding of the system performance, cost, and learning results.
- **Generic to models and datasets**. The platform should support different models and datasets for third-party developers, instead of customized for specific tasks.

2.2 Overview

FLASH Input. Besides the dataset and the model, a developer needs to specify the user behavior trace and extra FL configurations. The user behavior trace tracks the devices' status to simulate the behavior of clients in FL. Developers can use the default trace in FLASH (§2.3) or provide a new one. For FL configurations, we mainly follow the report of Google's large scale FL system [5, 57]: (1) ***Reporting deadline*** specifies how long the server waits for the selected clients to upload the model updates. (2) ***Goal client count*** specifies how many clients are selected to participate in the training each round. (3) ***Selection time window*** specifies how long the server waits in the selection stage. A round will be abandoned and restarted if the number of available clients is less than *goal client count*. (4) ***Maximum sample count*** is the max number of samples used for training on every single device.

Note that many of the above configurations are directly related to heterogeneity. For example, without heterogeneity, reporting deadline is not so significant as all clients run at the same speed synchronously. We will investigate the impacts of two critical configurations, i.e., reporting deadline and goal client count, and provide heuristic guidance on how to properly configure them in §4.

FLASH Output. Besides the common output provided by existing FL platforms, e.g., accuracy and communication cost, FLASH generates two new yet critical metrics: (1) ***Convergence time*** is the time taken to model convergence, i.e., when a model reaches a stable accuracy close to global optima. Note that the convergence time in FLASH is not the time of the simulation process, but the wall-clock time of running FL in the real world. (2) ***Client failure report*** includes the information on when and why a client fails to upload the model updates. We investigate the influence of heterogeneity on these metrics in §3.

As illustrated in Figure 1, running an FL task in FLASH mainly experiences two stages: pre-process and runtime.

- **Pre-processing Stage** (§2.4). FLASH processes the data that is required for taking into account heterogeneity at runtime. The *Training Profiler* profiles different devices' performance on the specified ML model under given settings.
- **Runtime Stage** (§2.5). FLASH performs heterogeneity-aware FL simulation with four components. *Configuration Parser* reads and parses the user's inputs and sets up the simulation environment. *Device Simulator* estimates different devices' training and communication time according to the profiling data. *User Behavior Simulator* checks whether a client is available for training and whether the client is able to upload its updates within the given time. *FL Process Simulator* follows the three-phase protocol to simulate the FL process.

Input Configurations	Leaf	PySyft	TFF	PaddleFL	FLASH
Non-I.I.D. dataset	✓	✓	✓	✓	✓
Goal Client Count	✓	✓	✓	✓	✓
Reporting Deadline	×	×	×	×	✓
Sample Num	×	×	×	×	✓

Output Metrics	Leaf	PySyft	TFF	PaddleFL	FLASH
Loss & Accuracy	✓	✓	✓	✓	✓
Convergence Round	✓	✓	✓	✓	✓
Communication Cost	✓	✓	×	×	✓
Convergence Time	×	×	×	×	✓
Failure Report	×	×	×	×	✓

Table 1: Compared to existing FL platforms, FLASH provides more practical metrics and configurations due to its heterogeneity-aware nature.

Field	Description	Example
user_id	Anonymized user id.	xxxyyyzzz
device_model	device type	SM-A300M
screen_trace	screen on or off	screen_on
screen_lock_trace	screen lock or unlock	screen_lock
time	time at current state	2020-01-29 05:52:16
network_trace	network condition	2G/3G/4G/5G/WiFi
battery_trace	battery charging state, battery level	battery_charged_off 96.0%

Table 2: Example of our user behavior trace

2.3 Contributed Dataset

To simulate FL in a heterogeneity-aware way, we collect data from large-scale real-world users. The data come from a popular input method app (IMA) that can be downloaded from Google Play[2]. The dataset covers 136k users and spans one week from January 31st to February 6th in 2020. The users mainly come from Indonesia, Columbia, and Brazil. Overall, they include 180 million trace items and 111GB data. The data can be divided into three different categories.

- **User behavior trace** tracks the device's meta information and its status changes, including battery charge, battery level, network environment, screen lock, and screen on and off. Table 2 shows an example of the collected trace.
- **Network performance** contains the network bandwidth between clients and the server across time. It is essential to simulate the model download/upload speed in FL.
- **M-Type input corpora** contain the input corpora from the users of our IMA from Columbia. Input word prediction

[2]For a double-blind review, we hide the exact app name. We will report the detailed information when the work is published.

is a typical use case of FL [16], while all existing open input datasets [9, 50] used by the research community are synthetic instead of directly obtained from IMA. Furthermore, M-Type shares the same user population with preceding behavior data, which enables us to jointly investigate the influence of heterogeneity.

Ethic considerations. All data are collected to improve user experience, with explicit agreements from users on user-term statements and strict policy in data collection, transmission, and storage. In addition, we take very careful steps to protect user privacy and preserve the ethics of research. First, our work is approved by the Research Ethical Committee of the institutes that the authors are currently affiliated with. Second, the usersâĂŹ identifies are all anonymized during the project. Third, the data are stored and processed on a private, HIPPA-compliant cloud server, with strict access authorized by the company that develops the IMA. The whole process is compliant with the public privacy policy of the company.

2.4 Pre-Processing Stage

Training Profiler is responsible for collecting the on-device training performance used to simulate devices' various training capabilities. An ML model is first transferred to a profiling service, where a desktop is connected to various mobile devices. The model will be tested on each device and the resource consumption (e.g., training time) will be profiled and retrieved back to the FL simulation platform. Currently, our profiling service includes three devices that represent different hardware capacities: Google Pixel 4 (high-end), Redmi Note 7 Pro (moderate), and Nexus 6 (low-end). We use DL4J [15], an open-source ML library, to implement the on-device training. Observing that the training time of the same model on the same device can also have high variance, we use a normal distribution to fit the training time during simulation instead of a fixed number. The design of our profiling service can facilitate the usage of FLASH for developers who have no access to various device types or are not skillful in mobile programming. The service can also be easily extended to more device types, on-device training libraries (e.g., TensorFlow [1]), and resource types (e.g., energy consumption and memory footprint).

2.5 Runtime Stage

Configuration Parser is to set up the simulation environment with given specifications. It first initializes virtual clients and the server in separate processes. A global model is loaded in the server and the data is loaded by clients. Each client is randomly assigned a user behavior trace that manipulates the device's status change over time. We observe that, at each round, online clients account for 4.9% to 15.7% of the total clients. As illustrated in Figure 2, we also observe a diurnal distribution of the percentage of online clients, which is consistent with the report of Google's FL system [5, 57]. Note that we choose to randomly assign behavior trace to generalize our trace to more datasets, but in practice there could be an inherent relationship between client data and user behavior [22]. We will validate our practice in §6.

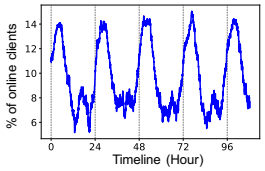

Figure 2: Percentage of online clients over time.

Device Simulator considers device heterogeneity in the simulation runtime. The behavior trace has recorded more than a thousand device models, making it difficult to precisely profile the training time for each device. Thus, Device Simulator maps the large number of real-world device models to the profiled devices. Here, we use the results of AI-Benchmark [19] as an intermediary. We first manually match top 792 device models to 296 devices categories that have been profiled by AI-Benchmark, e.g., "SM-A300M" to "Samsung Galaxy A3". The users of those device models almost account for 95% of the total population. We then map the AI-Benchmark devices to our profiled devices with the closest ranking. Finally, for the devices that AI-Benchmark does not profile, we randomly pick a profiled device category proportionally and assign it to the device. After the preceding "clustering" operation, each device model in the collected trace is mapped to a profiled device category, so its on-device training time is available.

User Behavior Simulator considers behavior heterogeneity in the simulation runtime. It checks whether a device is available for training (called online) at any time window based on the user's trace information. Here, we follow a common practice [5]: a device is online when it is idle, charged, and connected to an unmetered network (e.g., Wi-Fi).

The simulator also detects if a selected client fails to accomplish the task to upload the model updates, called *client failure*. FLASH categorizes a client failure to three possible causes: (1) ***Network failure*** is detected if it takes excessively long time (default: 3× the average) to communicate with the server due to slow or unreliable network connection. (2) ***Interruption failure*** is detected if the client fails to upload the model updates due to the user interruption, e.g., the device is uncharged during training. Note that FLASH supports suspend-resume of both training and network transmission as the device switches between online and offline. (3) ***training failure*** refers to the case when clients take too much time on training. §3.3 will further analyze such failure.

FL Process Simulator follows the classic three-phase protocol [5] to perform the heterogeneity-aware simulation. (1)

Data set	Femnist	Celeba	Reddit	Real-World
Task	Image Class.	Image Class.	Next Word Prediction	Next Word Prediction
Model	CNN	CNN	LSTM	LSTM
Total Clients	1,800	9,122	25,124	5,842
Goal Client Count	100	100	100	100
Selection Time Window	20s	20s	20s	20s
Reporting Deadline	310s	250s	90s	100s
Maximum Sample Count	340	30	50	100

Table 3: The default configurations for each task.

Each round in the selection phase, online clients check in with the server, and the server selects a portion of them for training. In the configuration phase, the model and the configuration are sent to selected clients where on-device training is performed. (2) In the reporting phase, clients report back the model update. (3) The server will update the global model if the fraction of reporting clients reaches the target value (default: 0.8). Currently, FLASH supports the following model aggregation algorithms: FedAvg (default) [32], q-FedAvg [28], and FedProx [26]. New aggregation algorithms can be freely integrated to FLASH as designed so.

2.6 Experimental Settings

Now we introduce the ML datasets and FL configurations used in our experiments.

Testing Data. We use three synthetic datasets commonly used in FL literature [3, 26, 28, 38], i.e. Reddit [44], Femnist [14], Celeba [36], and our real-world input corpus M-Type. Table 3 lists their meta information and the corresponding ML tasks. Each dataset is randomly split into a training set (80%) and a testing set (20%) before assigned to clients.

FL Configurations. Table 3 summarizes the default settings for each ML task. We mainly follow prior work [5, 16, 57] to set the default configurations in experiments. For the configurations introduced by heterogeneity as aforementioned in §2.2, we ran extensive microbenchmarks and heuristically choose a proper one so that the FL model achieves the best performance. §4 will further study how these configurations affect the FL process. If not specified otherwise, we use FedAvg as the default FL aggregation algorithm.

Simulation Environment. All our experiments are running on a Linux high-performance computing cluster with Red Hat Enterprise Linux Server release 7.3 (Maipo). The cluster has 10 GPU workers. Each worker is equipped with 2 Intel Xeon E5-2643 V4 processor, 256G main memory, and 2 NVIDIA Tesla P100 graphics cards. All the experiments cost more than 5,700 GPU-hours.

3 HETEROGENEITY'S IMPACTS ON FL PERFORMANCE

In this section, we explore whether and to what extent heterogeneity will affect the performance of FL.

3.1 Accuracy and Convergence Time

The accuracy and convergence time are two most critical metrics in FL. To measure the impact of heterogeneity on these two metrics, we conduct experiments on all aforementioned datasets using our platform. For each dataset, we run a heterogeneity-unaware version as well as a heterogeneity-aware version for comparison. Similar to prior work [32], we vary the number of local training epochs, a key setting in FL to balance the computation-communication cost.

We show the results in Figures 3 and 4. We summarize our observations and insights as follows.

• **Heterogeneity causes nontrivial accuracy drop in FL.** The accuracy drops by 2.3%, 0.5%, and 4% on Femnist, Celeba, and Reddit, respectively. The accuracy drops by 9.2% on M-Type, indicating that the negative impacts of heterogeneity are more severe in real-world settings. The primary reason of the resultant accuracy drop is that, when taking into account heterogeneity, different clients may have various participation levels, e.g., low-end devices can hardly participate and contribute their data. As a result, some of the in-active clients may be under-represented, which compromises the overall accuracy. Later in §3.2 we will further explore the issue of bias introduced by heterogeneity in FL.

• **Heterogeneity obviously slows down the FL convergence.** Compared to heterogeneity-unaware environment, the convergence time under heterogeneity is increased to 1.15× (Reddit with E = 1) and to 2.32× (Celeba with E = 20). The impact gets more distinct with larger local epochs, probably because too many local updates can affect the model convergence [26]. For Femnist and Celeba with 20 local epochs, the convergence time is even slowed down by around 12 hours. The reason is that, due to heterogeneity, the client may miss the deadline to report the model updates, resulting in less data involved in the training process. A training round may also fail if the server can not collect enough clients' updates to commit this round. §3.3 will dive deeper into the impacts of client failure and round failure.

• **Behavior heterogeneity is more influential than device heterogeneity.** The preceding results indicate the joint impacts from types of heterogeneity. To measure the impact of device heterogeneity and behavior heterogeneity respectively, we disable the device heterogeneity, i.e., all the clients have the same computation and communication capacity (with legend as "w/o device heter" in Figure 4). Similarly, we disable the behavior heterogeneity, i.e., clients are always

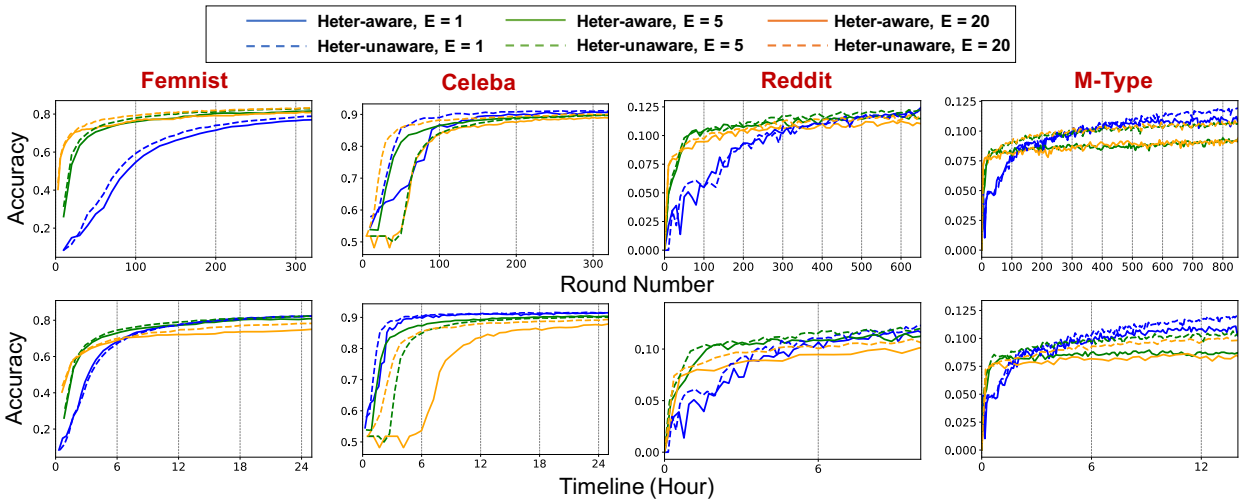

Figure 3: Heterogeneity decreases the model accuracy and slows down the convergence.

available at any time and will not be interrupted (with legend as "w/o behavior heter" in Figure 4).

As shown in Figure 4, behavior heterogeneity leads to a more significant accuracy drop than device heterogeneity, i.e., 9.5% vs. 0.4% on M-Type and 1.1% vs. 0.05% on Femnist, respectively (ratio). The reason is that, as we will show in §3.2, behavior heterogeneity introduces more participation bias and accuracy bias in FL compared to device heterogeneity. For convergence time, both device heterogeneity and behavior heterogeneity will slow down the convergence process while behavior heterogeneity is a bit more significant. For example, on Femnist dataset, the convergence time increases by 10.1 hours due to device heterogeneity and by 12.0 hours due to behavior heterogeneity. Both kinds of heterogeneity will introduce client failure to FL, as shown in §3.3.

Findings: Heterogeneity causes nontrivial performance degradation in FL, i.e., significant accuracy drop and convergence slowdown. Behavior heterogeneity is more significant for such performance degradation compared to device heterogeneity. It leads to two key implications. First, heterogeneity should not be ignored in FL development or simulation process, and a heterogeneity-aware platform like FLASH is required for developers to precisely understand how their model shall perform in real-world environment. Second, it urges researchers to develop more efficient FL protocol to relieve the negative impact of heterogeneity.

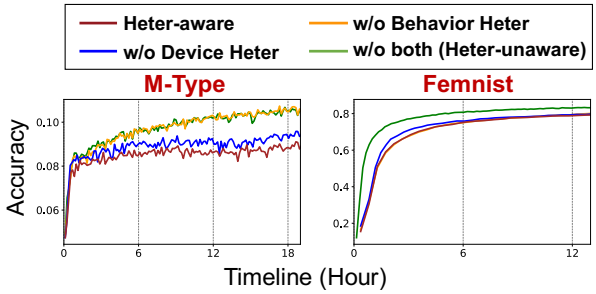

Figure 4: A breakdown of the impacts of different types of heterogeneity. Behavior heterogeneity causes more performance degradation than device heterogeneity. "Heter" is short for heterogeneity.

3.2 Bias of Participation and Accuracy

Bias in FL refers to the phenomenon that some clients receive more attention (e.g., higher accuracy or more participation) than others. Heterogeneity further magnifies the influence of bias as we will show in this section.

Definitions. In FL, due to the behavior heterogeneity, clients do not participate in the learning process with the same probability, which may lead to different levels of participation, i.e., participation bias. Due to the device heterogeneity, low-end devices are less likely to upload their updates to the global model, making them under-represented, i.e., accuracy bias. To measure the influence of the bias introduced by heterogeneity, we run the same FL tasks in §3.1 and record the related metrics, distribution of computation and accuracy. Similar to §3.1, we also break down to explore different types of heterogeneity's influence on bias.

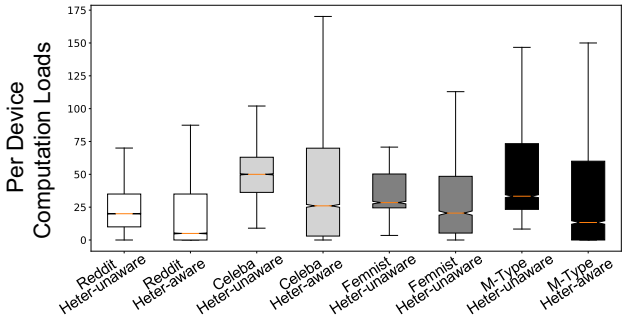

Figure 5: The distribution of computations across devices during FL training.

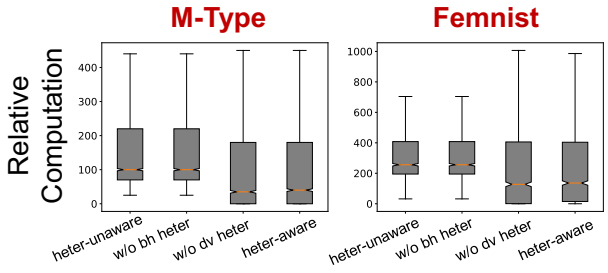

Figure 6: A breakdown of the impacts of different types of Heterogeneity on participation bias.

Participation bias. We take the amount of computation to reflect the participation degree of different clients and demonstrate the results in Figure 5 with the model converged. Since different models' absolute computation is hard to compare directly, we divide them by the amount of computation for a training epoch (noted as relative computation). We summary our findings as follows.

• The distribution gets more uneven. The variance is increased by 2.4× (Reddit) to 10.7× (Femnist). Compared to heterogeneity-unaware case where every client participates with an equal probability, the heterogeneity-aware case has a trend of polarization. On Celeba, the maximum computation load increases by 1.17×.

• The median computation drops by 28% (Femnist) to 75% (Reddit), indicating that the number of in-active clients increases significantly. Compared to the heterogeneity-unaware case where top 30% of the clients contribute 54% of the total computation, in heterogeneity-aware case they contribute 81% of the total computation, putting the in-active clients at a disadvantage.

• To investigate the reasons for these in-active clients, we inspected the number of involved clients over time and demonstrated in Figure 7. In the heterogeneity-unaware case, the involved clients accumulate quickly and soon cover the total population. While in heterogeneity-aware case, the accumulation speed gets much slower and it takes about 30× time to converge. In addition, about 5% of the total population does not participate in the learning process at all. We find that the model accuracy on these in-active clients decreases by 0.5% to 1.2%, similar to the accuracy drop in §3.1.

• As shown in Figure 6, behavior heterogeneity is the main reason for computation bias. It causes the similar computation distribution as the one in heterogeneity-aware case.

Accuracy bias. To measure the accuracy bias, we investigate the accuracy distribution in the total population. We demonstrate the results in Figure 8, from which we make the following key observations.

• Due to heterogeneity, more low-accuracy clients appear. Among the four datasets, M-Type is affected the most, with the drop of 12% (ratio) in terms of median accuracy. This indicates heterogeneity can introduce more bias on real-world dataset.

• The effect of bias becomes more obvious when we increase the number of training epochs. It shows that too many local updates (i.e., increase the number of training epochs) can obviously affect model's accuracy.

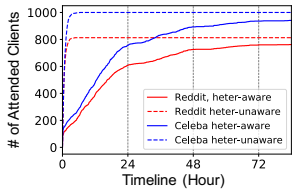

Figure 7: Attended client number over time.

• As shown in Figure 9, behavior heterogeneity introduces more low-accuracy clients where the median accuracy drops by 12.0% (ratio) on M-Type and by 0.5% (ratio) on Femnist. We do not detect a severe accuracy bias caused by device heterogeneity . It achieves a similar accuracy distribution as the heterogeneity-unaware case. It is probably because some of the low-end clients may still conform to the overall data distribution.

Findings: Heterogeneity introduces severe participation bias and accuracy bias to FL. It causes more in-active clients that seldom or never participate in the learning process, and also larger accuracy variation across clients. Behavior heterogeneity is mainly responsible for such bias. It urges researchers to mitigate the bias by designing a more "fair" FL system against heterogeneity.

3.3 Client Failure

Client failure refers to the phenomenon that a client misses the deadline to report the model updates in a round. The reasons of client failure include network failure, training failure, and interruption failure (refer to §2.5 for more details).

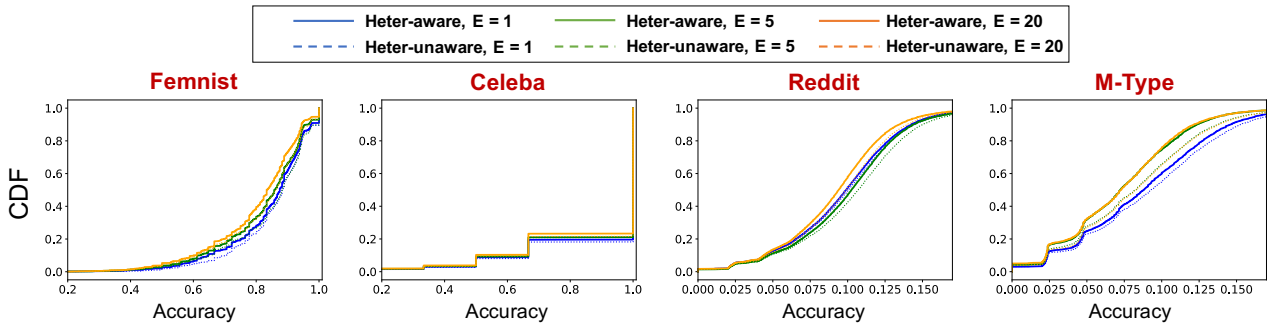

Figure 8: The accuracy distributions across clients, showing that heterogeneity amplifies the accuracy bias.

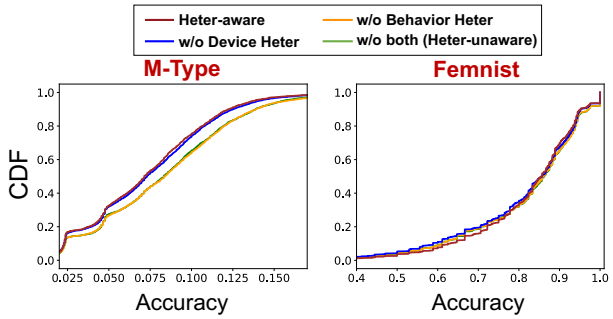

Figure 9: A breakdown of the impacts of different types of heterogeneity on accuracy bias.

Client failure is a unique challenge in FL system [26], as it slows down the model convergence and causes a waste of valuable device resources (computations, energy, etc.). However, client failure is seldom studied in prior work, probably because it is directly related to the FL heterogeneity.

To understand client failure, we zoom into the previous experiments on the Femnist and Reddit datasets under varied reporting deadlines. Similar to §3.1, we will also check device heterogeneity and behavior heterogeneity's influence on client failure. The key questions we want to answer here are: (1) how often the clients may fail and what are the corresponding reasons for these failures; (2) and which type of heterogeneity is the major factor. The results are illustrated in Figures 10 and 11, from which we make the following key observations.

Client failure is common under typical FL settings. The overall proportion of the failure clients reaches 10% on average, with an optimal deadline setting that achieves the shortest convergence time. A tight deadline increases the failure proportion because clients receive less time to finish their learning tasks.

• Network failure accounts for a small fraction of client failure (typically less than 2%) and it is more stable than the other types of failure.

• Training failure is heavily affected by the deadline. Such a failure can account for the majority of the client failures when the deadline is set too tight. Even with the optimal deadline setting, such failure still occurs due to the randomness, i.e., the training speed can be influenced by other background tasks (§ 2.5).

• Interruption failure is also affected by the deadline but in a more moderate way. We further breakdown the interruption failure into three sub-categories corresponding to three restrictions on training [5]. The results show that, the training process is interrupted by user interaction, battery charged-off, and network changes with a probability of 46.06%, 36.96%, and 17.78% respectively.

Device heterogeneity causes more failure clients than behavior heterogeneity. According to Figure 11, we can find that device heterogeneity is more responsible for the client failure. For example, on M-Type, device heterogeneity causes 14% failure clients while behavior heterogeneity causes 2.5%. Training failure is mainly caused by device heterogeneity because it introduces weak devices that suffer longer training time. Interruption failure is caused by behavior heterogeneity because user behavior is introduced that may interrupt the training and communication progress.

Findings: Heterogeneity leads to considerable client failures. For each round, around 10% of the selected clients fail to participate in the model update under typical FL settings. It slows down the model convergence and leads to wasted hardware resource consumption of clients. Device heterogeneity is the major reason for client failure while behavior heterogeneity also introduce around 2.5% failures. To mitigate the impacts of such failure, one may explore the opportunities of dynamic round deadline [24] and smart client selection [38].

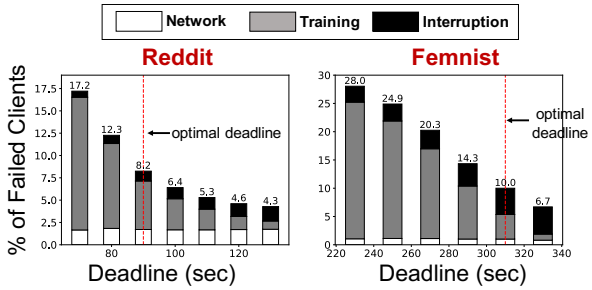

Figure 10: The prevalence of different failure reasons. The optimal deadline (red line) refers to the one that achieves the shortest convergence time.

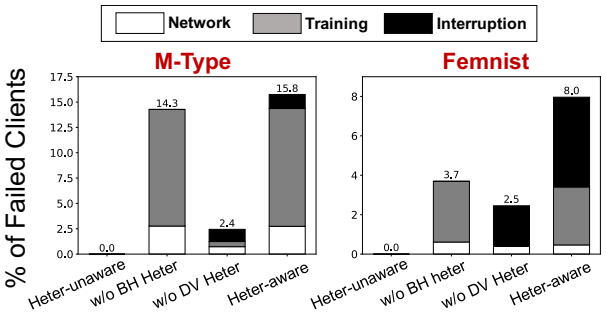

Figure 11: Different kinds of heterogeneity's influence on client failure.

4 CONFIGURING FL SYSTEMS

As shown in §2.6, many FL configurations are introduced due to the heterogeneity. How to select these configurations, however, has been seldom studied in the previous work due to the lack of heterogeneity-aware FL platform along with representative dataset. In this section, we show how these configurations impose nontrivial impacts on the performance of the FL process. For each configuration, we mainly report two critical metrics in FL (as in §3.1): convergence accuracy and convergence time.

4.1 Reporting Deadline

Reporting deadline is set to avoid excessively long server waiting time in each round of FL process. The clients that cannot upload their model updates within the deadline will be discarded, i.e., client failure. Reporting deadline is inherently related to heterogeneity, since the heterogeneity is the main source of client failure as previously shown in § 3.3. In existing FL literature, the usage of reporting deadline and its influences are mostly ignored [9, 28] or considered without in-depth exploration [27, 38] due to the lack of support on FL heterogeneity.

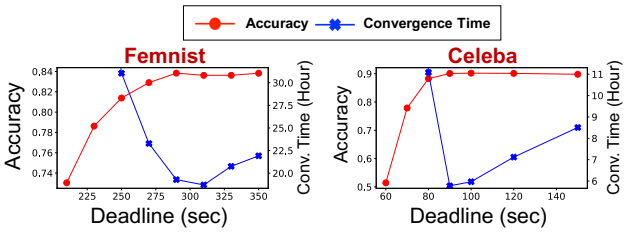

Figure 12: The impacts of reporting deadline on convergence time and accuracy.

The key questions we want to answer here include: (1) how reporting deadline affects the FL performance, and (2) how FL developers can select an adequate reporting deadline. To this end, we run experiments on Reddit and Femnist datasets across varied reporting deadlines. We fix other configurations as their default values (same as the ones reported by Google [57]). We summarize our observations as following.

How reporting deadline affects convergence accuracy. As illustrated in Figure 12, with the reporting deadline increases, the model's convergence accuracy increases at first, and then remains stable. Taking Femnist as an example, the accuracy raises from 0.74 to 0.84 as the deadline increases from 120 seconds to 290 seconds, and keeps rather stable around 0.83–0.84 with even higher reporting deadline. The reason of such a trend is that when the deadline is too tight (within a short time period), a large portion of clients, especially those low-end devices, can hardly (or seldom) contribute to the global training process (as discussed in § 3.2), which leads to high accuracy bias and the unsatisfactory mean accuracy across all clients. while the deadline is too loose (in a long time period), more clients are involved in FL training but the model's capacity has been reached.

How reporting deadline affects convergence time. As shown in Figure 12, with the increase of reporting deadline, the convergence time first drops and then rises up, indicating that there is a potential optimal deadline setting (i.e., shortest convergence time) in the middle. Taking Celeba as an example, the convergence time drops from 11 hours to 5.8 hours as the deadline increases from 80 seconds to 90 seconds, and rises gradually to 8.5 hours when the deadline increases to 150 seconds. For each round, the deadline controls the round time and how many clients (or data) can contribute to the model convergence. When the deadline is tight, it is important to wait longer to have more clients upload their model update. While the deadline reaches a certain value (the optimal point), further increasing the deadline does not help as the time is wasted on waiting for a few more clients.

How to select a good reporting deadline. According to Figure 12, there exists an optimal deadline that reaches the highest accuracy with shortest convergence time (e.g., 310s

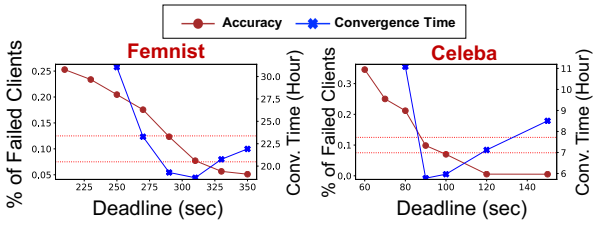

Figure 13: The optimal reporting deadline, which achieves the fastest convergence, can be set when the client failure rate is around 10%.

for Femnist and 90s for Celeba). Given its high impacts, however, how to choose a proper deadline has rarely been studied in FL literature. Thus, we present a heuristic approach on how to choose the optimal reporting deadline. The proposed approach is based on an observation: the optimal deadline is correlated to the client failure rate (§2.6). As shown in Figure 13, for all tested datasets, the convergence time reaches the optimal (shortest) when the client failure rate is around 10%. This finding guides developers to select a proper deadline based on the monitored client failure rate with only one or several rounds. It is much more efficient compared to exploring different deadlines with end-to-end experiments (as our experiments did), each of which can take hundreds of rounds. The saved efforts (e.g., experiment time in GPU-hours) can be up to two to three orders of magnitude as estimated.

Findings: Choosing an optimal reporting deadline is critical to FL performance, and it exists at a middle point that can be obtained by monitoring the client failure rate. A tight reporting deadline compromises the model's accuracy while a loose reporting deadline slows down the convergence. A proper reporting deadline can be tuned when the client failure rate is around 10%. Developers can monitor this metric to dynamically tune the reporting deadline during FL training.

4.2 Goal Client Count

Goal client count refers to the number of selected clients in each round. Due to the heterogeneity, the clients may not always stay available during the learning process. Our trace reveals the proportion of online clients (4.9% ∼ 15.7% in Figure 2). Even though, the absolute number of online clients is still large for a popular application with millions of users. Existing work [9, 21, 27, 28] simply chose tens of or at most hundreds of clients without elaborating the rationales.

The key questions we seek to answer here are: (1) how goal client count affects the FL performance, and (2) how to select a proper goal client count. To this end, we run experiments on four testing datasets with various goal client counts (from 10 to 1,000) and fix other FL configurations to their default value (§2.6). We summarize our observations as following.

How goal client count affects convergence accuracy. As illustrated in Figure 14, a small goal client count (e.g., 25) results in high accuracy fluctuation, or even no convergence. For such cases, even if the best model is taken, its accuracy still drops observably compared to a relatively large goal client count (e.g., 3.9% (ratio) for M-Type). However, when the goal client count reaches 100, further increasing it has very marginal effect on accuracy improvement. Taking Reddit as an example, the accuracy increases by only 0.4% (ratio) when we increase the goal client count from 100 to 200.

How goal client count affects convergence time. According to Figure 14, on most datasets except Celeba, increasing the goal client count since 25 cannot accelerate the convergence process effectively, even though more data are involved in cross-device parallel training. Prior work also noticed this diminishing acceleration effect with more selected clients [5, 32]. The reason is that the current aggregation algorithm *FedAvg* cannot effectively utilize the uploaded gradients in a scalable way.

How to choose an appropriate goal client count. Since the benefits of increasing goal client count are rather limited, it is wise to choose an adequate one to save network bandwidth. We highlight the optimal goal client counts that achieve the highest accuracy in Figure 14. Empirically, a value between 100 and 300 is recommended.

Notably, existing FL literature seem to "unreasonably" use a low goal client count, presumably to minimize the experiment cost [21, 27, 28]. For example, *Leaf* [9] selects only 5 clients each round in experiments and the reported accuracy is 0.747 [9], while our experiments indicate that the accuracy can achieve 0.809 with a higher goal client count (100).

Findings: On basis of all benchmark datasets widely used in FL literature, it is strongly recommended to select $100 - 300$ **clients per round to obtain maximal learning performance while not wasting client resources.** It contradicts to some prior knowledge [9, 21, 27, 28], where much fewer clients are selected on the same datasets as we use. What is more, the benefits of increasing the goal client count are rather limited in terms of accuracy and convergence time. It urges researchers to optimize the FL algorithm's degree of parallelism.

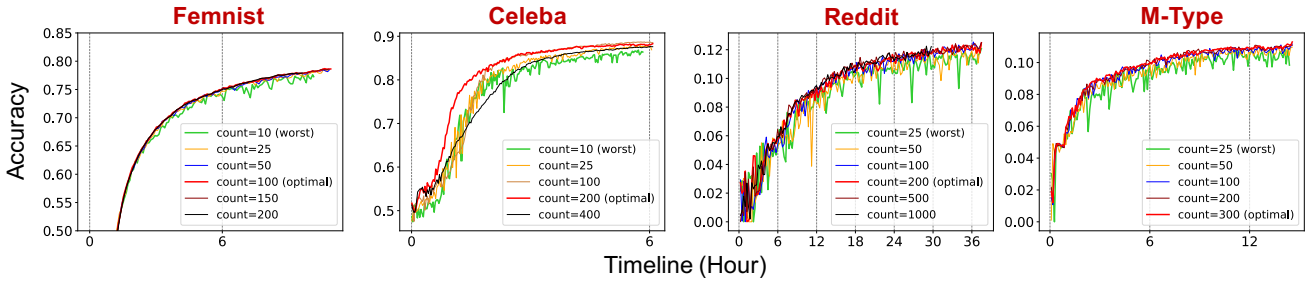

Figure 14: The test accuracy over time under various goal client counts. A small count leads to lower or unstable convergence accuracy, while a too large count can not further speed up the convergence.

5 HETEROGENEITY'S IMPACTS ON FL OPTIMIZATIONS

Optimizations have been proposed atop federated learning [23, 24, 26, 28, 32, 35, 39, 43]. Some traditional optimizations in distributed ML can also be applied to FL. Those techniques, however, have not been evaluated when taking into account heterogeneity. Hence, we study how heterogeneity affects the applicability of existing techniques. Specifically, we focus on two typical directions: gradients compression and model aggregation algorithms.

5.1 Gradients Compression

Since client-server communication is often reported as a major bottleneck in FL [22], we first investigate the gradients compression algorithms that are extensively studied to reduce the communication cost. We choose three popular algorithms: *Structured Update* [23], *Gradient Dropping* (*GDrop*) [2], and *SignSGD* [4]. We use Femnist and M-Type with their default configuration settings (§2.6). We empirically determine the optimal hyper-parameters for each algorithm through massive experiments: for *Structured Update*, we set the max rank of the decomposited matrix to 100; for *GDrop*, we set the weights dropout threshold to 0.005; for *SignSGD*, we set learning rate lr to 0.001, momentum constant β to 0, and weight decay λ to 0. A "no compression" version is also included as the baseline. Table 4 summarizes the results.

Heterogeneity has few impacts on gradients compression algorithms. We measure the accuracy change introduced by heterogeneity (noted as *Acc. Change* in Table 4). A bit surprisingly, we observe that the accuracy degradation introduced is similar to the one that we find in §3.1. On average, the accuracy drops by 1.7% (ratio) on Femnist and 5.3% (ratio) on M-Type. It is reasonable because heterogeneity will not affect the compressed gradients thus will not lead to a further accuracy drop.

The performance across different algorithms vary a lot. On *Structured Update*, a negligible accuracy decrease (typically less than 1%) is observed compared to the baseline without any compression. Sometimes, it even slightly outperforms the baseline (Femnist), because the reduced communication time can make more clients upload their updates. Using *SignSGD* and *GDrop* results in unstable model accuracy. For example, *SignSGD* achieves higher accuracy than other algorithms on M-Type (2.9%-6.0% better than the baseline), but underperforms other algorithms on Femnist. *GDrop* suffers a severe accuracy drop (10.9%-18.0%) on M-Type while it achieves the best compression ratio (0.1%-2.1%).

Gradients compression algorithms cannot speed up the model convergence. Although all these algorithms compress the gradients and reduce the communication cost significantly (the compression ratio reaches from 0.1% to 39.4%), the convergence time is seldom shortened (only structured update shortens the convergence time to 0.93× at most) and lengthened in most cases. For example, on M-Type under heterogeneity-aware environment, the convergence time is lengthened by 1.3× to 2.5× for all compression algorithms. There are two reasons. First, we find that communication accounts for only a small portion of the convergence time compared to on-device training. Most devices can finish the downloading and uploading in less than 30 seconds for a model around 50M while spending more time (1-5 minutes with 5 epochs) on training. Second, the accuracy increases slowly when the gradients are compressed, thus taking more rounds to converge.

Such observations set FL apart from traditional distributed ML [11, 25, 49], where the compression can both reduce communication cost and end-to-end training time. Given that FL typically operates on unmetered network, the only advantage of gradients compression, i.e., reducing communication cost, becomes dispensable to some extent. As a result, it becomes questionable whether gradients compression is still needed in FL tasks.

Dataset	Algo.	Acc (%) Heter-unaware	Acc (%) Heter-aware	Acc Change (ratio)	Convergence Time Heter-unaware	Convergence Time Heter-aware	Compression Ratio
Femnist	No Compression	84.1 (0.0%)	83.0 (0.0%)	1.2%↓	5.56 hours (1.0×)	5.96 hours (1.0×)	100%
	Structured Update	**84.2** (0.1%↑)	**83.2** (0.3%↑)	1.1%↓	**5.23 hours (0.95×)**	**5.56 hours (0.93×)**	6.68%
	GDrop	82.2 (2.2%↓)	81.5 (1.8%↓)	0.8%↓	7.17 hours (1.3×)	7.98 hours (1.3×)	21.4%∼28.2%
	SignSGD	79.0 (6.1%↓)	76.3 (8.1%↓)	3.4%↓	7.62 hours (1.4×)	20.5 hours (3.4×)	**3.125%**
M-Type	No Compression	9.86 (0.0%)	9.28% (0.0%)	5.9%↓	0.54 hours (1.0×)	1.23 hours (1.0×)	100%
	Structured Update	9.93 (0.6%↑)	9.08 (2.2%↓)	8.6%↓	**0.53 hours (0.98×)**	**1.59 hours (1.3×)**	39.4%
	GDrop	8.09 (18.0%↓)	8.27 (10.9%↓)	2.2%↑	5.34 hours (10.0×)	4.29 hours (3.5×)	**0.1%∼2.1%**
	SignSGD	**10.4** (6.0%↑)	**9.55** (2.9%↑)	8.5%↓	1.45 hours (2.7×)	3.93 hours (3.2×)	3.125%

Table 4: The performance of different gradients compression algorithms. Numbers in the brackets indicate the accuracy change (ratio) compared to the "No Compression" baseline. "Acc. Change" refers to the accuracy change introduced by heterogeneity. Compression ratio is the fraction of compressed gradients size to the original size.

Findings: Gradient compression algorithms can not speed up the model convergence. What is worse, it could slow down the convergence process although it can significantly compact the uploaded gradients per round. It reminds researchers that new protocols to reduce the communication cost should be validated in an end-to-end way instead of only focusing on per-round results.

5.2 Aggregation Algorithms

Aggregation algorithm is a key component in FL that determines how to aggregate the updates (gradients) uploaded from multiple clients. Besides *FedAvg*, various aggregation algorithms are proposed to improve efficiency [27, 38, 39], ensure fairness [28], preserve privacy [6, 39], etc. To study how heterogeneity affects these algorithms, we choose two representative ones: *q-FedAvg* [28] and *FedProx* [27], both of which are open-sourced. We choose *FedAvg* as the baseline, and run experiments on Femnist and Reddit. Due to their different optimization goals (*q-FedAvg for addressing fairness issues* and *FedProx* for improving efficiency), we make the comparison separately.

Q-FedAvg loses its effectiveness towards better fairness after heterogeneity is introduced. To address the fairness issues in FL, *q-FedAvg* minimizes an aggregate *reweighted* loss parameterized such that the devices with higher loss are given higher relative weight. We use the learning rate and batch size reported in § 3.1 for both algorithms. We randomly sample 100 clients from online clients each round, set the reporting deadline as the optimal value according to §4.1. We use the same metrics (variance, worst 10% accuracy, and best 10% accuracy) to evaluate the algorithms' effectiveness.

The results are summarized in Table 5, showing that, without considering heterogeneity, *q-FedAvg* achieves higher worst 10% accuracy than *FedAvg* and obtains lower cross-clients variance on both datasets. However, when heterogeneity is considered, the variance reduction decreases from 26.3% to 21.7% on Femnist and from 10.5% to 3.7% on M-Type. It is

Dataset	Hete.	Algo.	Average	Worst 10%	Best 10%	Var. $\times 10^{-4}$
Femnist	Unaware	FedAvg	82.13%	61.1%	**97.2%**	213
		q-FedAvg	**82.66%**	**64.7%**	95.1%	**157** (26.3% ↓)
	Aware	FedAvg	81.22%	61.1%	94.9%	203
		q-FedAvg	**81.24%**	**64.7%**	**95.1%**	**159** (21.7% ↓)
M-Type	Unaware	FedAvg	**8.15%**	2.33%	**13.5%**	19
		q-FedAvg	7.78%	**2.33%**	13.0%	**17** (10.5% ↓)
	Aware	FedAvg	7.47%	2.27%	12.3%	16.2
		q-FedAvg	**7.47%**	**2.33%**	**12.4%**	**15.6** (3.7% ↓)

Table 5: Test accuracy for *q-FedAvg* and *FedAvg*. "Var" represents the accuracy variance across clients.

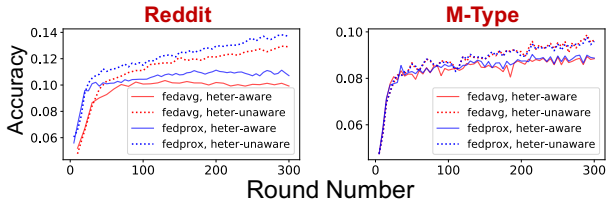

Figure 15: The training performance of FedProx and FedAvg with and without heterogeneity.

because heterogeneity introduces severe bias (§3.2), which makes *q-FedAvg* less effective in ensuring fairness.

FedProx is less effective in improving the training process with heterogeneity considered. *FedProx* is proposed to tackle with device heterogeneity in federated networks. Compared to *FedAvg*, *FedProx* tolerates partial work, where devices can perform various amount of work based on their available system resources, while *FedAvg* simply drops the stragglers. *FedProx* also adds a proximal term to the local optimization objective (loss function) to limit the impact of variable local updates.

The results are summarized in Figure 15. *FedProx* converges much quicker than *FedAvg* (50 rounds in our case) due to its tolerance of partial work. It also obtains a 6% accuracy increase compared to *FedAvg*. However, due to the heterogeneity, the accuracy drops by 27%. On M-Type, *FedProx* only slightly outperforms *FedAvg*, and the heterogeneity causes

an accuracy drop of 7.5%. Note that *FedProx* incorporates device heterogeneity into its design while leaving behavior heterogeneity unsolved. We manually check the attended clients and find that only 51.3% clients have attended the training when the model converges. So the model may have been dominated by these active clients thus may perform badly on other clients.

Findings: The effectiveness of aggregation algorithms can be undermined by heterogeneity. In our experiment, *q-FedAvg* algorithm fails to obtain the same improvements, i.e, accuracy variance reduction, as that in heterogeneity-unaware environment; *FedProx* algorithm can accelerate the convergence process but the accuracy drop caused by heterogeneity remains significant. This urges researchers to take heterogeneity into account when they explore FL optimization techniques.

6 DISCUSSION

Ground truth. Similar to prior FL platforms, FLASH executes FL tasks in a simulation way. FLASH is carefully designed to simulate the real-world deployment by considering the heterogeneity. However, we realize that a gap may still exist for unexpected FL glitches, e.g., software failure. We plan to further validate FLASH with real-world deployment. Nevertheless, the observed patterns from FLASH, e.g., client availability and failure proportion, are consistent with the results reported from a large-scale FL deployment by Google [5]. The findings made by this work are still valid.

Trace Selection Bias. The user trace datasets are collected from our IMA's users (app-specific) who mainly reside in three countries (geo-specific). The traces may not be representative enough to other FL scenarios. However, we believe that our findings are still meaningful because (1) FL task is always app-specific and improving IMA experience is a key scenario of FL [5, 16, 57]; (2) our dataset is large enough to cover the general behavior patterns of smartphone users, which is consistent with prior study as aforementioned. Furthermore, new user traces can be seamlessly plugged into FLASH where researchers can reproduce all experiments mentioned in this paper, so that they can validate whether and how much heterogeneity matters in their scenarios.

Validity of decoupling. In real world, the heterogeneity is inherently coupled with the non-iid data distribution [22]. FLASH decouples the heterogeneity from the data distribution, i.e., randomly assigning a user behavior trace to each client, to generalize our trace to other datasets. We use M-Type to verify such design because it shares the same user population with our trace. According to Figure 16, the gap between the coupled case and the decoupled case is trivial compared to the gap between the heterogeneity-unaware case and heterogeneity-aware case. It justifies the design of FLASH of decoupling heterogeneity from any third-party datasets.

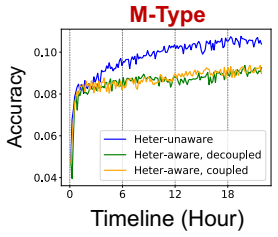

Figure 16: Decoupling is verified on M-Type.

7 RELATED WORK

Federated Learning (FL) is an emerging privacy-preserving learning paradigm that distributes training on decentralized devices such as smartphones holding local data samples [25, 32]. FL has been increasingly popular these days, deployed on large-scale users to enhance their keyboard functionality [5, 8] and improve virtual keyboard search suggestion quality [57]. There have been some FL platforms [9, 40, 47, 50, 56], most of which run in a simulation way, given the high cost of real-world deployment on a large number of users. Those platforms, however, fail to incorporate behavior heterogeneity and device heterogeneity into their design. As a result, there exists a gap between those platforms and real-world deployment. Recognizing such gap, we build the first heterogeneity-aware FL platform FLASH (§2).

FL optimizations have been proposed atop origin FL algorithm. The research directions include reducing the communication cost between server and clients [7, 13, 23, 31, 46, 48], further enhancing the privacy guarantee [3, 6, 33, 34, 37], ensuring better fairness across clients [21, 28, 35], and minimizing the on-client energy cost [24]. Those optimizations, however, have never been evaluated in a heterogeneity-aware environment, making their benefits unclear in real-world deployment. Our experiments in §5 demonstrate that heterogeneity indeed has considerable impacts on those optimizations, advocating that new FL techniques need to be justified on a heterogeneity-aware platform.

Heterogeneity in distributed system has been studied by the system community for years. Distributed machine learning (e.g., using parameter server [25]) is a common way to solve large-scale machine learning tasks. In this paradigm, multiple workers train a global model collaboratively similar to FL. The hardware heterogeneity may cause the whole system inevitably dragged down by slower workers. Various systems and algorithms have been proposed to tackle this problem [11, 12, 17, 18, 20, 42, 45]. In FL, however, heterogeneity not only resides in clients' hardware capacity, but also user behavior. As we will experimentally show in §3, the latter often has more significant impacts on the training

process. Thus, those prior efforts cannot be applied to solve the heterogeneity challenge in FL tasks.

8 CONCLUSION

We have built FLASH, the first heterogeneity-aware platform for federated learning. Based on FLASH, We perform extensive experiments to first anatomize the impacts of heterogeneity, which shows that (1) heterogeneity causes non-trivial performance degradation in FL tasks, up to 9.2% accuracy drop and 2.32× convergence slowdown; (2) a misconfiguration of FL system parameter can significantly degrade the FL performance; (3) common FL optimizations can be compromised and rethought with heterogeneity considered. These results suggest that heterogeneity should be taken into consideration in further research work and that optimizations to mitigate the negative impacts of heterogeneity are promising.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Alham Fikri Aji and Kenneth Heafield. 2017. Sparse Communication for Distributed Gradient Descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, 440–445. https://doi.org/10.18653/v1/d17-1045

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2018. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459* (2018).

[4] Jeremy Bernstein, Jiawei Zhao, Kamyar Azizzadenesheli, and Anima Anandkumar. 2019. signSGD with Majority Vote is Communication Efficient and Fault Tolerant. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=BJxhijAcY7

[5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046* (2019).

[6] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.

[7] Keith Bonawitz, Fariborz Salehi, Jakub Konečnỳ, Brendan McMahan, and Marco Gruteser. 2019. Federated learning with autotuned communication-efficient secure aggregation. *arXiv preprint arXiv:1912.00131* (2019).

[8] Daniel Ramage Brendan McMahan. 2017. Federated Learning: Collaborative Machine Learning without Centralized Training Data. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html. (2017). Published April 6, 2017.

[9] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).

[10] Zheng Chai, Hannan Fayyaz, Zeshan Fayyaz, Ali Anwar, Yi Zhou, Nathalie Baracaldo, Heiko Ludwig, and Yue Cheng. 2019. Towards Taming the Resource and Data Heterogeneity in Federated Learning. In *2019 USENIX Conference on Operational Machine Learning, OpML 2019, Santa Clara, CA, USA, May 20, 2019*, Bharath Ramsundar and Nisha Talagala (Eds.). USENIX Association, 19–21. https://www.usenix.org/conference/opml19/presentation/chai

[11] Chia-Yu Chen, Jungwook Choi, Daniel Brand, Ankur Agrawal, Wei Zhang, and Kailash Gopalakrishnan. 2018. AdaComp : Adaptive Residual Gradient Compression for Data-Parallel Distributed Training. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 2827–2835. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16859

[12] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. 2016. Revisiting Distributed Synchronous SGD. *CoRR* abs/1604.00981 (2016). arXiv:1604.00981 http://arxiv.org/abs/1604.00981

[13] Yang Chen, Xiaoyan Sun, and Yaochu Jin. 2019. Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation. *IEEE Transactions on Neural Networks and Learning Systems* (2019).

[14] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. 2017. EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2921–2926.

[15] Eclipse. 2020. Deep Learning for Java. https://deeplearning4j.org/. (2020). Accessed Mar 16, 2020.

[16] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).

[17] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*. 1223–1231.

[18] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, Aditya Akella and Jon Howell (Eds.). USENIX Association, 629–647. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh

[19] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 0–0.

[20] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware Distributed Parameter Servers. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 463–478. https://doi.org/10.1145/3035918.3035933

[21] Yihan Jiang, Jakub Konečnỳ, Keith Rush, and Sreeram Kannan. 2019. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488* (2019).

[22] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open

problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[23] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[24] Li Li, Haoyi Xiong, Jun Wang, Cheng-Zhong Xu, and Zhishan Guo. [n. d.]. SmartPC: Hierarchical Pace Control in Real-Time Federated Learning System. ([n. d.]).

[25] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, Jason Flinn and Hank Levy (Eds.). USENIX Association, 583–598. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[26] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2019. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873* (2019).

[27] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2018. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127* (2018).

[28] Tian Li, Maziar Sanjabi, and Virginia Smith. 2019. Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497* (2019).

[29] Xuanzhe Liu, Huoran Li, Xuan Lu, Tao Xie, Qiaozhu Mei, Feng Feng, and Hong Mei. 2017. Understanding diverse usage patterns from large-scale appstore-service profiles. *IEEE Transactions on Software Engineering* 44, 4 (2017), 384–411.

[30] Xuan Lu, Xuanzhe Liu, Huoran Li, Tao Xie, Qiaozhu Mei, Dan Hao, Gang Huang, and Feng Feng. 2016. PRADA: Prioritizing android devices for apps by mining large-scale usage data. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 3–13.

[31] Hugh Brendan McMahan, Dave Morris Bacon, Jakub Konecny, and Xinnan Yu. 2019. Communication Efficient Federated Learning. (Nov. 7 2019). US Patent App. 16/335,695.

[32] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

[33] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963* (2017).

[34] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 691–706.

[35] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. 2019. Agnostic federated learning. *arXiv preprint arXiv:1902.00146* (2019).

[36] The Chinese University of Hong Kong Multimedia Laboratory. 2020. Large-scale CelebFaces Attributes (CelebA) Dataset. http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html. (2020). Accessed May 22, 2020.

[37] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *arXiv preprint arXiv:1812.00910* (2018).

[38] Takayuki Nishio and Ryo Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.

[39] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. 2019. Secure federated submodel learning. *arXiv preprint arXiv:1911.02254* (2019).

[40] PaddlePaddle. 2020. Federated Deep Learning in PaddlePaddle. https://github.com/PaddlePaddle/PaddleFL. (2020). Accessed Jan 28, 2020.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.

[42] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distributed Comput.* 69, 2 (2009), 117–124. https://doi.org/10.1016/j.jpdc.2008.09.002

[43] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. 2019. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445* (2019).

[44] PushShift.io. 2020. Reddit Dataset. https://files.pushshift.io/reddit/. (2020). Accessed May 22, 2020.

[45] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*. 693–701.

[46] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. 2019. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. *arXiv preprint arXiv:1909.13014* (2019).

[47] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. 2018. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017* (2018).

[48] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated multi-task learning. In *Advances in Neural Information Processing Systems*. 4424–4434.

[49] Ananda Theertha Suresh, Felix X Yu, Sanjiv Kumar, and H Brendan McMahan. 2017. Distributed mean estimation with limited communication. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3329–3337.

[50] Tensorflow. 2020. TensorFlow Federated: Machine Learning on Decentralized Data. https://www.tensorflow.org/federated. (2020). Accessed Jan 28, 2020.

[51] Wikipedia. 2020. California Consumer Privacy Act. https://en.wikipedia.org/wiki/California_Consumer_Privacy_Act. (2020). Accessed Feb 5, 2020.

[52] Wikipedia. 2020. China Internet Security Law. https://en.wikipedia.org/wiki/China_Internet_Security_Law. (2020). Accessed Feb 5, 2020.

[53] Wikipedia. 2020. FacebookâĂŞCambridge Analytica data scandal. https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal. (2020). Accessed Feb 5, 2020.

[54] Wikipedia. 2020. General Data Protection Regulation. https://en.wikipedia.org/wiki/General_Data_Protection_Regulation. (2020). Accessed Feb 5, 2020.

[55] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*. 2125–2136.

[56] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.

[57] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903* (2018).