Towards Mitigating Device Heterogeneity in Federated Learning via Adaptive Model Quantization

Ahmed M. Abdelmoniem
KAUST
Saudi Arabia

Marco Canini
KAUST
Saudi Arabia

Abstract

Federated learning (FL) is increasingly becoming the norm for training models over distributed and private datasets. Major service providers rely on FL to improve services such as text auto-completion, virtual keyboards, and item recommendations. Nonetheless, training models with FL in practice requires significant amount of time (days or even weeks) because FL tasks execute in highly heterogeneous environments where devices only have widespread yet limited computing capabilities and network connectivity conditions.

In this paper, we focus on mitigating the extent of device heterogeneity, which is a main contributing factor to training time in FL. We propose *AQFL*, a simple and practical approach leveraging adaptive model quantization to homogenize the computing resources of the clients. We evaluate *AQFL* on five common FL benchmarks. The results show that, in heterogeneous settings, *AQFL* obtains nearly the same quality and fairness of the model trained in homogeneous settings.

CCS Concepts: • **Computer systems organization → Client-server architectures**; • **Computing methodologies → Neural networks**; **Distributed algorithms**.

Keywords: Federated Learning, Heterogeneity, Model Quantization

ACM Reference Format:
Ahmed M. Abdelmoniem and Marco Canini. 2021. Towards Mitigating Device Heterogeneity in Federated Learning via Adaptive Model Quantization. In *The 1st Workshop on Machine Learning and Systems (EuroMLSys '21), April 26, 2021, Online, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3437984.3458839

EuroMLSys '21, April 26, 2021, Online, United Kingdom

ACM ISBN 978-1-4503-8298-4/21/04…$15.00
https://doi.org/10.1145/3437984.3458839

1 Introduction

Exploiting machine learning (ML) techniques for big data analytics and building predictive models has become predominant in many applications [12, 16, 41]. Traditionally, training of deep neural networks (DNNs) is performed over centrally-collected datasets with little to no regard for privacy concerns. However, with the growth of connected devices, large datasets are nowadays distributed at end-user devices and data cannot be centrally collected due to privacy concerns. Moreover, various regulations have been imposed for the preservation of consumer and user privacy, such as GDPR [1] and CCPA [2]. Consequently, DNN training is moving to the data-rich network edge, including through outsourced computations over a widely heterogeneous collection of mobile devices and other distributed training architectures [10, 14, 29].

A popular paradigm for distributed collaborative learning is Federated Learning (FL) [7, 22, 29]. In FL, client devices collaboratively learn a global model with the help of a central server without transmitting the private data [31]. This approach has recently drawn much attention from industry (e.g., [7, 17, 36, 41]) and academia (e.g., [9, 39]). FL, however, faces major challenges that hinders its wide adoption in real applications [7, 40, 41]. One of the main challenges is the heterogeneity of clients (in terms of hardware capabilities and/or connectivity) participating in the training process. Device heterogeneity can have a large impact on the quality of the trained models [5] and this is the focus of our work.[1] Device heterogeneity arises as population bias during training due to uneven client sampling as slower or poorly connected devices are more susceptible to failures or missing the deadline for reporting module updates [7].

To demonstrate the extent of device heterogeneity's impact on model quality, consider the following scenarios:

- An ideal HOmogeneous case (**HO**): where devices have homogeneous computational and network access capabilities (thus, all devices finish within the deadline) and devices remain available throughout training (thus, devices are sampled uniformly at random).
- A realistic Device Heterogeneous case (**DH**): where devices have heterogeneous hardware and network

[1] We account for data heterogeneity by using non-IID datasets in our experiments. Behavioral heterogeneity, which is left for future work, may also impact the model quality (esp. w.r.t. fairness) as it affects the availability of the clients throughout training [5].

Table 1. Summary of the benchmarks used in this work.

Task	ML technique	Model	Dataset	Model Size [bytes]	Total Clients	Selection Count (**sc**)	Reporting Deadline (**ddl**) [*s*]	Maximum Sample Count	Learning Rate	Quality metric
Next Word Prediction	RNN	LSTM	Reddit [33]	24,722,496	813	100	27	50	0.5	12.83%
			Shakespeare [29]	3,271,488	1,129	50	142	50	0.8	46.97%
Image Classification	CNN	2 Conv2D Layers	FEMNIST [11]	26,414,840	3,400	100	60	340	0.01	80.098%
		1 Conv2D Layer	CelebA [28]	124,808	9,343	100	15	30	0.01	90.6%
Cluster Identification	Traditional ML	Logistic Regression	Synthetic [9]	2,400	9,367	50	23	340	0.005	84.21%

access characteristics while client availability also remains uniform.

Following the methodology from our work in [5], we perform experiments across a range of FL configurations where we vary a number of hyper-parameters that influence the model quality. Figure 1 shows the average test accuracy normalized to the **HO** baseline and contrasts the two scenarios. The box-plots for the two cases show the average test accuracy measured for experiments sweeping a wide range of hyper-parameters settings of five different FL benchmarks (listed in Table 1). The results show that device heterogeneity has a significant impact on model quality. Specifically, at the median (shown in green), the normalized average test accuracy of **DH** (see the figure caption) is 0.91× that of **HO**. Moreover, the accuracy across experiments in **DH** varies widely, and in the worst case training does not converge.

In this paper, we take a first step towards mitigating the unwanted effects of device heterogeneity on the quality and fairness (i.e., how model accuracy varies across clients) of DNN models trained in FL environments. To this end, we propose an adaptive model quantization technique that varies the quantization level in proportion to the resources available at clients' end-devices. This approach reduces both the computational and communication overhead of the training process, which results in significantly better quality and fairness of the trained models. We support our claims via empirical experiments on five common FL benchmarks (see Table 1) used in FL literature. These benchmarks span different ML tasks including image and facial features recognition, next word prediction and cluster identification.

In summary, our contributions are:

- We show the effects of device heterogeneity on the training process in heterogeneous FL environments.
- We highlight the benefits of quantized model training and its minimal impact on convergence in FL settings.
- We propose Adaptive Quantized Federated Learning (*AQFL*), a simple and practical approach based on model quantization to mitigate the model quality and fairness degradation.
- Via experimentation on common FL benchmarks, we show the benefits of *AQFL* in homogenizing the devices which helps in retaining the quality and fairness of the ideal case.

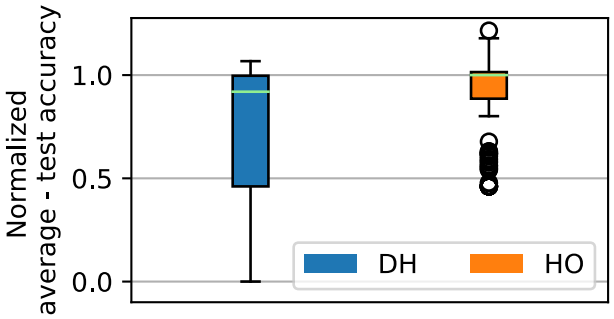

Figure 1. Impact of device heterogeneity on model quality. The test accuracy is averaged over all devices and normalized by the baseline accuracy of each benchmark.

2 Background

Due to the growing computational power of users' end-devices, coupled with concerns over transmitting private information, it is increasingly attractive to store data locally and push network computation to the edge. A commonly used architecture for federated learning setting is depicted in Figure 2. In this setting, N clients owning different sets of data, which has common data structure, learn a global model via a centralized aggregation FL server. To train a model, the following steps are performed [29]:

1. The clients check-in with the FL server, and then the server typically selects a sample of clients for training and pushes a copy of the up-to-date global model.
2. The clients perform an equal number of local optimization steps as determined by task designer.
3. Clients encrypt all or a subset of the updated local model parameters with encryption [32], differential privacy [3], or secret sharing [8] techniques and push the encrypted model to the FL server.
4. The FL server performs secure aggregation of the local models pushed by the clients.
5. The FL server updates the state of the global model.

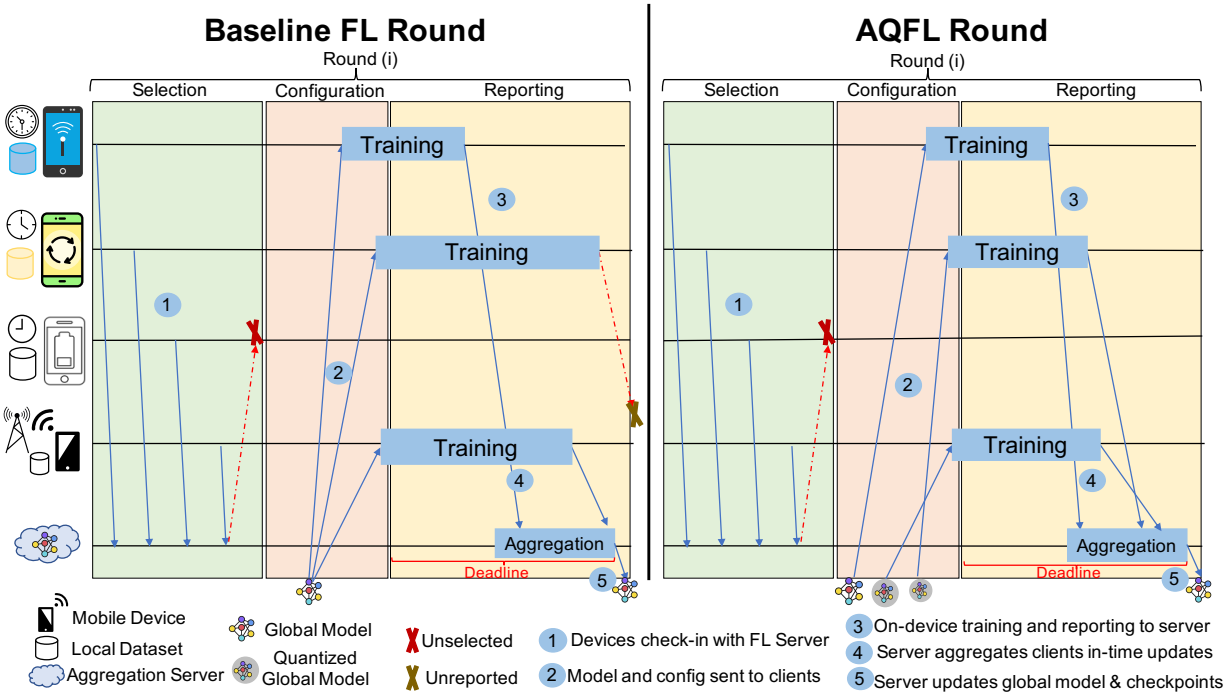

Figure 2. Phases of Training FL model in a resource heterogeneous environment

The central server, which performs aggregation and hosts the global model, invokes the above process frequently and continues so until the model accuracy or loss function converges to a target value. This setting does not depend on the specific ML algorithms in use (e.g., logistic regression, SVM, or DNN). Moreover, the incentive for client participation is that they will share and benefit from the deployment of the converged model in their applications.

2.1 System Aspects of the FL Life Cycle

FL production systems consist of different players that take role in the FL life cycle. The main players are the FL server and the devices which perform an FL task on their local data. The server is typically a cloud-based distributed server which performs the aggregation and collects some metrics about the task. An FL task is a computation (either training or evaluation) for a specific FL population. The task performs either training on the local dataset with given hyper-parameters, or evaluation of the global model on a held-out validation dataset to obtain some performance metrics. An FL population is uniquely identified by the learning problem, or application for which the model is trained or evaluated [7, 17, 41].

The process starts when clients that are ready to run an FL task for a given FL population join the system. Then, the server, within a certain time window, selects a subset of clients from the online clients that joined within the time window. Typically, the server selects a few hundred from the (tens of) thousands online clients to participate in the FL task. FL tasks are broken into rounds and a round is concluded when certain stages complete successfully during which clients are expected to stay connected throughout their duration. In each round, the server drives the stages towards the completion of the round. The main stages in FL systems are selection, configuration, and reporting [7]:

Selection: The server waits for clients to join within a time window and if the target number of clients becomes available, the server proceeds to the configuration stage; otherwise this stage is restarted later.

Configuration: The server configures the aggregation processes (either single or distributed instances) for the selected devices. The aggregation algorithm can be either a simple Federated averaging [29] or a variant which exploits secure aggregation [8]. Then, the server proceeds with sending an FL task and a checkpoint of the global model to each client. Then, the clients start executing the FL task on their local datasets.

Reporting: The server waits until a time limit or deadline, for the participating clients to upload their model updates. Then, the server aggregates the updates received before the deadline using the configured aggregation mechanism. This stage and hence the round is deemed successful when the server updates the global model. This happens if the number of received updates before the deadline matches a configured target update fraction; otherwise the round fails and the received updates are ignored, in which case the round is restarted from scratch.

2.2 Device Heterogeneity Challenge for FL

For FL to be widely adopted, the impact of device heterogeneity on model quality should be mitigated. To this end, the training of DNNs on resource-constrained mobile and embedded devices should be efficient. As a result, various works proposed model compression for this purpose. Specifically, quantized DNNs have drawn a significant attention. A quantized DNN uses discrete values to represent both weights and activations of the model. Quantized DNNs have several benefits due to their smaller bit-width and smaller model size in that they: 1) can increase efficiency by using fixed-point computations; 2) can speed up the computation and reduce memory usage on resource-limited devices; 3) can speed up the upload and download time of the model. It is reported that a speedup of 2× to 3× is obtained for quantized inference compared to float, with almost 10× speedup with DSP or NPU units [19, 23].

It is also shown that, for the same time budget, quantized training of DNNs (e.g., bit-widths ≥ 8) can obtain accuracy on par or even higher than its floating-point counterpart [20]. However, for the same number of iterations, quantized DNNs may result in noticeable accuracy degradation (e.g., for bit-widths ≤ 4)[13, 18]. Yet, quantizated training of DNN models on mobile devices in FL setting can be beneficial to mitigate the resource variations in device heterogeneous settings. So, quantization can be seen as a homogenizing tool of the computation and network of the clients. Next, we present *AQFL*, a practical scheme for enabling efficient training of FL models.

3 Adaptive Quantized Federated Learning

In real-world end-device settings, different bit-widths are supported by different devices [19, 20, 23, 27, 30]. This hardware flexibility allows for the ability of configuring the model with different quantization levels to match the variety of hardware configurations of clients' mobile or IoT devices. Therefore, it is natural to rethink the design of the FL server and allow the flexibility of choosing for the clients among different quantized versions of the model. The server can then make a per-client decision and send the model version that best matches the capabilities of the selected clients in each round. To this end, the server would maintain a single version of the model in floating-point format and quantize the model during the configuration phase of each round. Alternatively, the FL server may use Quantizable DNNs, which are a special type of quantized DNNs that can flexibly adjust the bit-width on the fly, i.e., turning on different bit mode by applying different quantization levels [13].

System Design: We leverage model quantization and design *AQFL* system to homogenize the devices participating in the training in a client-agnostic manner. Fig. 2 contrasts the design of *AQFL* with the existing FL system. They are nearly identical except that the *AQFL* system imposes a minor change to the operations of the FL server during the configuration phase in which the server needs to decide on whether to send a quantized model and its precision.

Algorithm 1 AQFL Algorithm

Input: $K, T, \eta, E, w^0, N, k = 1, \ldots, N$
for $t = 0, \ldots, T-1$ **do**
  K online clients check-in with the server.
  Server selects, at random, a subset S_t of K devices
  Server collects information from selected clients to decide the quantization level Q_k for each client k
  Server sends a custom quantized $Q_k(w^t)$ to device k
  parfor $k = 0, \ldots, K-1$ **do**
    run SGD for E epochs on F_k with step-size η
    a new version of the model is produced $Q_k(w_k^{t+1})$
    send the updated model $Q_k(w_k^{t+1})$) to the server
  end parfor
  Server updates the model: $w^{t+1} = \frac{1}{K} \sum_{k \in S_t} Q_k^{'}(w_k^{t+1})$
end for

Quantization Decision: The decision of the model precision is quite challenging in highly heterogeneous environments. One feasible approach is that, during the selection stage, the server sends a reference (or dummy) model to each selected client. The dummy model is an unused replica of the global model being trained. Then, the client profiles forward and backward pass(es) and sends its computational specs and profile run times to the server. The server, which maintains computational profile(s) of the dummy model in different precision for reference device(s), can map the client profiles to a reference profile that will likely allow the device to finish training and send updates within the deadline. Alternatively, the server uses established benchmarks [6, 19] and quantizes the model to a precision that likely meets the deadline. We leave further exploration of these ideas to future work.

3.1 *AQFL* Algorithm

The *AQFL* algorithm is depicted in Algorithm 1. First, the clients check-in with the server and the server samples at random a subset S_t from the available clients. Then, the server collects the clients' computational profiles to aid in making the decision of the per-client model quantization precision Q_k. The precision Q_k is chosen to likely meet the time budget (or the deadline) at minimal impact on convergence.[2] Then, during the configuration phase, the server sends the custom quantized model $Q_k(w^t)$ to each client selected to participate in the training. Then, the clients train using the quantized model and upload the model updates to the server. Finally, the server de-quantizes the per-client model updates

[2] As we show the impact of quantization varies based on the benchmark. However, for reasonable quantization-levels (≥ 4), the impact is negligible.

and aggregates them to update the global model, which is typically stored in floating-point precision.

3.2 Practical Challenges

Realizing *AQFL* in practice faces several challenges, which we now highlight along with a discussion of possible solutions. A full exploration of these is part of our ongoing work.

Device quantization support: how can the FL server identify the range of supported bit-widths on the device for making per-device choices of the quantized model? This can be hard to achieve with tight privacy constraints. To overcome this, some of the clients may voluntarily share the meta-data about either their device model or the range of supported bit-widths when they check in with the FL server. The server could then maintain an association table with different hardware targets and theirs supported bit-widths.

Dynamic client environment: how can the FL server adapt its quantization decision for each device when device workload and network conditions vary? This can be hard to achieve with no knowledge about the changing dynamics of device workload and network conditions on the client side. To resolve this, the server could try to learn and model the environment of the clients to predict the optimal quantization level. For instance, the server could employ online reinforcement-learning to build an adaptive decision model for the quantization level.

Relying on the server may suffer from low quality decisions due to the limited availability of and/or outdated information. Alternatively, the system could rely on client-side mechanisms to make the decisions which provides higher quality decisions but at the cost of increased overhead on the client. For instance, the FL server could send several bit-width versions of the quantized model to the client and the client caches them locally. Then, the client could monitor its own workload and network conditions and adaptively alternate between the quantized models. For example, DC2 [4] monitors the network conditions and adjusts the control knob of the communication reduction methods adaptively.

4 Preliminary Evaluation

Our experiments address the following questions:

- What is the impact on model quality and fairness from using quantized models in an ideal FL setting?
- Is there any benefit from employing AQFL in various resource heterogeneous FL settings?

4.1 Relevant Metrics

We collect the average test accuracy and loss of the test clients as indicators of model's quality in each testing round.[3] We collect the same metrics for the training clients in training rounds. We collect several fairness metrics: 1) Cosine similarity between the test (and train) accuracy distributions and all-one vectors to measure the uniformity of the distribution (larger is better); 2) KL divergence of the test (and train) accuracy distributions to measure the entropy (larger is better); 3) Jain's Fairness Index of test (and train) accuracy [21]. We, however, only present test accuracy and cosine similarity as representative of the other metrics.

Benchmarks: We use five benchmarks covering a variety of FL applications used in several prior works [5, 7, 9, 17, 24, 25, 29, 39]. Table 1 summarizes the application, dataset, and default configuration of each benchmark. Note that the datasets are partitioned among the clients in a non-IID manner, which represents the most common scenario in FL settings. The benchmarks use common configurations as follows: *batch size* is 10, *number of epochs* is 1, *min selection count* is 10, *update fraction* is 0.8 (80%), selection time window is 20 seconds, and local optimizer is SGD.

Platform: We run experiments using the HeterFL simulation environment [5] on a GPU cluster. HeterFL is an extension of Flash [39], which simulates wall-clock accurate execution times of FL tasks. Flash also multiplexes the training of many clients onto a single GPU. Flash follows a real-world trace and maps the devices into three hardware categories: high-end (HE), moderate (M), and low-end (LE) devices. Their computational profiles are close to three real-world hardware specs. Finally, the per-device network upload and download speeds are randomly assigned from 20 different distributions representing real-world networks.

Heterogeneity: We conduct the experiments in the following two ideal and realistic scenarios. **HO** – The homogeneous case where clients are always available and have homogeneous hardware and link speed configurations. **DH** – The heterogeneous case where clients are always available but both their device type and link speed are sampled at random. The device partitions per round are random and on average over the full training the percentages of selected LE, M, and HE devices are 55%, 44%, and 1%, respectively.

Implementation We implement quantization using the built-in quantization API of TensorFlow [35, 36]. To quantize the model, we make use of the quantization function in the *tensorflow.contrib.quantize* library.[4] The quantization function updates the model computation graph with the required quantization operations for the chosen bit-width. Specifically, it takes as input the graph, and the quantization bits for the weights and biases of the model. Note that the procedure inserts fake quantization operations to simulate the error introduced by quantization.

4.2 Homogeneous Scenario with Quantization

We present the results of training using floating-point and quantized models in the ideal **HO** scenario. We evaluate

[3] Testing rounds are invoked every 1% of the total rounds and test clients are sampled from the FL population with a maximum cap of 3,500 clients.

[4] While most TensorFlow versions support quantization, we implement and evaluate *AQFL* using TensorFlow v1.14.

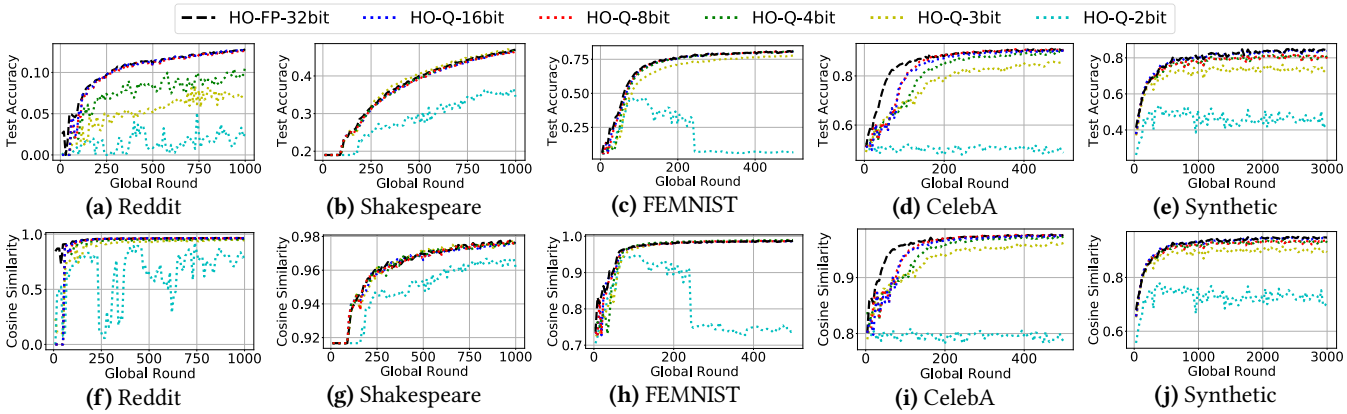

Figure 3. Model quality and fairness of FL tasks using quantized models in **HO**. The plots show the test accuracy (top row) and cosine-similarity (bottom row) on the y-axis and global training rounds on the x-axis.

the test accuracy and cosine-similarity of the model over the training rounds. We use 5 commonly used quantization levels (i.e., 16, 8, 4, 3, 2-bits) [13, 20].

Figure 3 shows that, for all benchmarks, quantization with precision of 16-bit and 8-bit quantization results in nearly the same average and cosine-similarity of test accuracy as of the floating-point baseline. This shows that not only the quality and fairness of moderately quantized models are acceptable, but they can used to reduce the run-time of the training for ill-capable (or slow) devices. The impact of qunatization for CelebA is noticeable (esp., for up to the first 150 rounds). This could be attributed to the batch normalization employed in CelebA's model architecture, which requires alternative quantization strategies [26, 38]. We observe also that 4-bit quantization results in lower but acceptable model quality and fairness for most benchmarks, except for Reddit, where there is a noticeable impact on the average model quality.

We note that, for extreme quantization levels (less than 4 bits), most benchmarks converge to a significantly lower test accuracy and achieve worse fairness among clients. This implies that the server either should avoid using such low quantization levels or use them only when the pool of available clients are limited to extremely low-capability devices. These results suggest that settling on a globally acceptable quantization level is hard to achieve and therefore there is a need for adapting the quantization level to the available resources of the target hardware.

4.3 Heterogeneous Scenario

We present the results of the realistic resource-heterogeneous scenario (**DH**) and evaluate whether *AQFL* is able improve the model quality to retain or nearly achieve the same model quality as in **HO** case. We evaluate the test accuracy and cosine-similarity of the model over the training rounds. We compare the homogeneous case (**HO**) and the heterogeneous case (**DH**) with and without *AQFL*. Figure 4 shows that, for all benchmarks, the **DH** case without *AQFL* compared to the **HO** case results in significant degradation in the average and cosine-similarity of test accuracy by up to 25% and 0.5%, respectively. This shows that **DH** can hinder the FL task and may result in a model with unsatisfactory quality and fairness. As shown in the last row of Figure 4, the degradation for **DH** without *AQFL* is because on average nearly $\approx 20\%$ of the selected clients are not successful in uploading their updates. These clients are mostly the ones with limited resources (i.e., LE devices), which miss the deadline and hence their updates are not incorporated into the global model.

In contrast, the results show that *AQFL* in the **DH** case helps to mitigate the impact from heterogeneity. We observe that both the average test accuracy (quality) and cosine-similarity (fairness) of the model are restored to values similar to those obtained in the ideal **HO** case. This is because the *AQFL*-enabled server picks the right-sized quantized model based on the device's computational class, which reduces the training run time for slow devices. As shown in the last row of Figure 4, the number of successful clients for **DH** with *AQFL* increases by 12-20% and approaches 100% participation (e.g., for Shakespeare and CelebA, the client success rate is at 100%). Note that the red line of **DH** with AQFL overlaps the black line for **HO** for Shakespeare and CelebA. Even though the client success rate with *AQFL* is at 100% for CelebA, we observe that all cases reach to same accuracy at the end, but *AQFL* has worse model quality up until round 150. This is because of the impact from quantization on batch normalization as detailed in §4.2.

5 Related Work

Federated Learning (FL): is a new learning approach for privacy-preserving learning on distributed datasets. In FL, the training is performed, with the help of a central server, on decentralized devices which frequently produce new data samples. The main requirement is that data does not leave

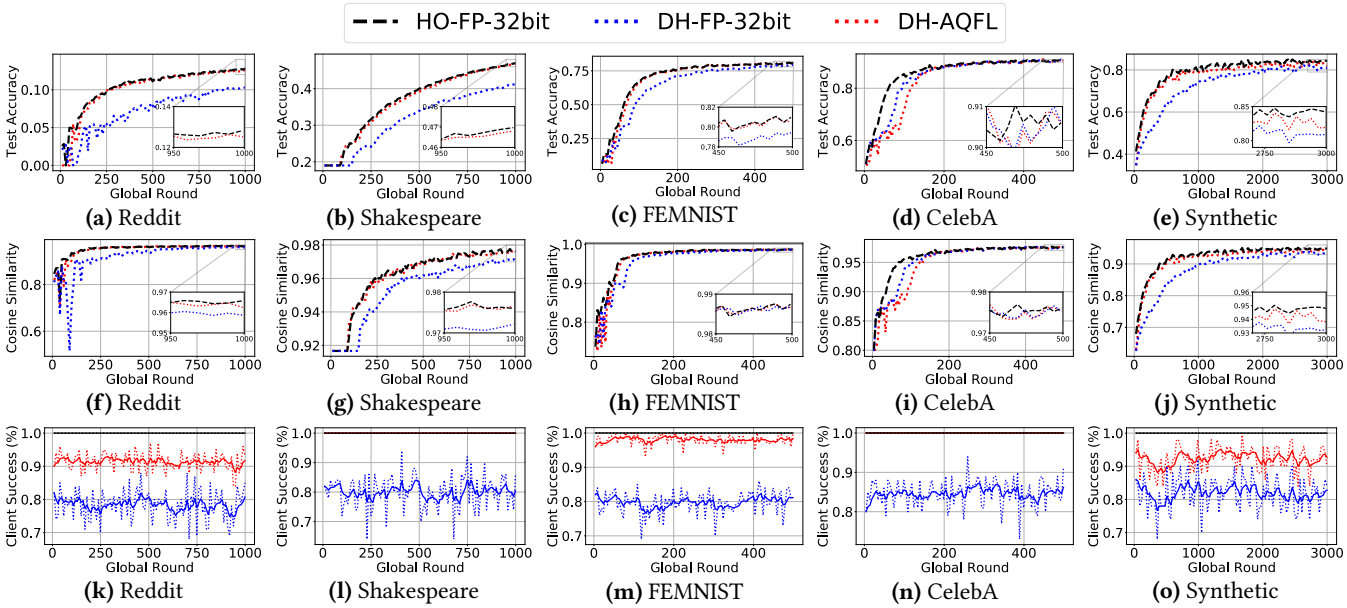

Figure 4. Model quality and fairness of FL tasks in **HO**, **DH**, and **DH** with *AQFL*. The plots show test Accuracy (top-row), cosine-similarity (middle-row), and Client success rate (bottom-row) on y-axis and training round on x-axis.

the local storage of the data source [29]. Therefore, FL has gained popularity and is currently deployed for a large number of users to enhance the functionality of virtual keyboards (e.g., the search suggestion [7, 41]). Several works developed FL frameworks for simulating FL settings [5, 9, 36, 39]. HeterFL [5] is a simulation platform that reflects the heterogeneity settings in FL and provides various evaluation metrics to assess the quality and fairness of the trained models. In this work, we extended HeterFL with the *AQFL* algorithm and use it to conduct the experiments.

System heterogeneity: Heterogeneity is one of the major challenges for distributed systems. In FL, device heterogeneity results in model quality degradation due to stragglers (i.e., slow workers) that slow down the training process [5, 7]. In the FL setting, the heterogeneity is not limited to the heterogeneity in device capabilities. Specifically, data distribution among clients, client sampling method, and user behavior are other sources of heterogeneity in FL scenarios [7]. Several works tried to address this problem via algorithmic solutions [24, 25], however their practical effectiveness is questionable [5, 39]. In contrast, this work proposes quantization as an effective and practical solution to the problem, showing promising results.

Quantized DNN training: Quantized DNNs map both weights and activations to discrete spaces to achieve smaller models and faster computations [13, 15, 20]. Quantized training is classified into post-quantization [20, 30] or quantization-aware [18, 27] training. Post-quantization training methods cannot support less than 8-bit values as the model quality significantly degrades for $\leq$ 4-bit quantization. Quantization-aware training adapts to the quantization noise during training to be able to support lower precision. These methods are fixed-bit, and changes in bit-width during training is not permitted. Notably, modern ML frameworks support model quantizaton (e.g., TensorFlow [35, 37] and PyTorch [34]).

6 Conclusion

We showed the impact from device heterogeneity on the quality and fairness of the FL trained models. To mitigate the impact, we proposed *AQFL*, a simple and practical FL system design which homogenizes the capabilities of the devices. *AQFL* leverages the readily available support for quantized training on mobile and IoT devices. Our evaluation, which spans five common FL benchmarks, showed that moderate model quantization imposes minimal impact on the model quality and fairness. *AQFL* quantizes the model on a per-client basis, which helps in achieving nearly the same quality and fairness of the models trained in homogeneous environments.

We believe our work will benefit: (1) Practitioners, who design FL applications deployed in real-world environments and do not want to worry about heterogeneity and its impact on model quality and fairness; (2) Researchers, who will use our system as the basis for introducing more enhancements that address other types of heterogeneity. As part of our future work, we plan to investigate the challenges that hinders the deployment of *AQFL* in practice and other approaches that help in mitigating the impact of heterogeneity in FL settings.

References

[1] 2016. General Data Protection Regulation. https://en.wikipedia.org/wiki/General_Data_Protection_Regulation

[2] 2018. California Consumer Privacy Act. https://en.wikipedia.org/wiki/California_Consumer_Privacy_Act

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *CCS*.

[4] Ahmed M. Abdelmoniem and Marco Canini. 2021. DC2: Delay-aware Compression Control for Distributed Machine Learning. In *INFOCOM*.

[5] Ahmed M. Abdelmoniem, Chen-Yu Ho, Pantelis Papageorgiou, Muhammad Bilal, and Marco Canini. 2021. On the Impact of Device and Behavioral Heterogeneity in Federated Learning. arXiv:2102.07500 [cs.LG]

[6] AI Benchmark. 2021. Performance Ranking. https://ai-benchmark.com/ranking.html

[7] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *MLSys*.

[8] 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 *CCS*.

[9] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. LEAF: A Benchmark for Federated Settings. arXiv:1812.01097

[10] Kapil Chandorikar. 2019. Introduction to Federated Learning and Privacy Preservation. https://towardsdatascience.com/introduction-to-federated-learning-and-privacy-preservation-75644686b559.

[11] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. 2017. EMNIST: Extending MNIST to handwritten letters. In *IJCNN*.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei Fei Li. 2009. ImageNet: a Large-Scale Hierarchical Image Database. In *CVPR*.

[13] Kunyuan Du, Ya Zhang, and Haibing Guan. 2020. From Quantized DNNs to Quantizable DNNs. arXiv:2004.05284 [cs.CV]

[14] Nils Gruschka, Vasileios Mavroeidis, Kamer Vishi, and Meiko Jensen. 2018. Privacy Issues and Data Protection in Big Data: A Case Study Analysis under GDPR. *Big Data* (2018).

[15] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

[16] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep Speech: Scaling up end-to-end speech recognition.

[17] 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:1811.03604

[18] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *Journal of Machine Learning Research* 18 (2017), 6869–6898.

[19] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. 2019. AI Benchmark: All About Deep Learning on Smartphones in 2019. arXiv:1910.06663 [cs.PF]

[20] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *IEEE CVPR*.

[21] Raj Jain, Arjan Durresi, and Gojko Babic. 1999. Throughput fairness index: An explanation. *ATM Forum contribution* 99, 45 (1999).

[22] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. In *Workshop on Private Multi-Party Machine Learning - NeurIPS*.

[23] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv:1806.08342 [cs.LG]

[24] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *MLSys*.

[25] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. 2020. Fair Resource Allocation in Federated Learning. In *ICLR*.

[26] Dachao Lin, Peiqin Sun, Guangzeng Xie, Shuchang Zhou, and Zhihua Zhang. 2020. Optimal Quantization for Batch Normalization in Neural Network Deployments and Beyond. arXiv:2008.13128 [cs.LG]

[27] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards Accurate Binary Convolutional Neural Network. In *NeurIPS*.

[28] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *ICCV*.

[29] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*.

[30] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-Free Quantization Through Weight Equalization and Bias Correction. In *IEEE/CVF ICCV*.

[31] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE Symposium on Security and Privacy (SP)*.

[32] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* (2018). https://doi.org/10.1109/TIFS.2017.2787987

[33] Pushshift. [n.d.]. Reddit Datasets. https://files.pushshift.io/reddit/

[34] pytorch.org. 2020. QUANTIZATION. https://pytorch.org/docs/stable/quantization.html

[35] tensorflow.org. 2020. Module: tf.quantization. https://www.tensorflow.org/api_docs/python/tf/quantization

[36] tensorflow.org. 2020. TensorFlow Federated: Machine Learning on Decentralized Data. https://www.tensorflow.org/federated

[37] tensorflow.org. 2020. TensorFlow Lite 8-bit quantization specification. https://www.tensorflow.org/lite/performance/quantization_spec

[38] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. 2018. Training and Inference with Integers in Deep Neural Networks. arXiv:1802.04680 [cs.LG]

[39] Chengxu Yang, QiPeng Wang, Mengwei Xu, Shangguang Wang, Kaigui Bian, and Xuanzhe Liu. 2020. Heterogeneity-Aware Federated Learning. arXiv:2006.06983

[40] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology* 10, 2 (2019).

[41] 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:1812.02903