Design and Analysis of a GDPR-Compliant Federated Machine Learning System

Nam Do
Brown University

Sam Thomas
Brown University

Abstract

We design and implement a federated machine learning model that demonstrates different degrees of GDPR compliance. In so doing, we introduce two different axes of compliance - namely *frequency* and *degree* of GDPR compliance - and measure the impact of different permutations of the two. We demonstrate that, even under the strictest models of GDPR compliance, the impact of GDPR compliance only causes an additional .25x increase in time and no significant increase in memory required per thread to implement GDPR compliance under simple learning benchmarks.

1 Introduction

Efficiency and privacy are often opposing forces for developers when building application code. That is, in order to ensure that user's online behavior can be kept sufficiently private, certain procedures need to be put into place that may add storage or performance overheads. These overheads are expensive to developers, as storage overheads necessitate more storage hardware and performance overheads generally result in dissatisfied users. However, ever more frequent is the enactment of privacy legislation, which require certain levels of privacy compliance in applications.

The European Union's General Data Protection Regulation (GDPR) (add citation) has brought this conversation into the forefront, as it is one of the first major pieces of privacy legislation that has had major ramifications on large companies behaviors and their products. The GDPR provides users a number of rights against companies as it relates to their data, and two such rights are:

1. The right to have any subset of their data updated.
2. The right to have any subset of their data deleted.

These rights have given applications users, via their Data Protection Agency, a means to directly challenge a company's or product's practices as it relates to their data. Furthermore, based on prior litigation (add citation), user data has been broadly defined to include "inferred data" about a user. That is, if a user provides a set of data to an application and the application processes that data in such a way that the application infers more data about the user, a user still has the right to modify or delete this data.

Such is the motivation for exploring the impact that GDPR compliance has on machine learning models. Machine learning inherently infers information based on a set of provided information. As such, the groundwork is laid for a potential scenario in which GDPR compliance can hamper the performance or storage required for a machine learning-dependent application.

One application design model that is becoming more commonly deployed to address user privacy is to store sensitive user data in the user's local application (D3PT [9], and others). From here, some form of local computation is performed before a result is sent for central aggregation in the central server. The benefit of such a model is that the central server can perform tasks while being completely agnostic to raw local data about any individuals *and* the issue of storing large sums of user data is pushed to the user's device as opposed to a central database.

Applied to machine learning, this design model is referred to as *Federated Machine Learning*. At a high level, as described above, training is performed on locally on user data on their devices, and the result of this training is then aggregated centrally. From here, the central server has a working machine learning model while being completely agnostic to user data. However, as we have described above, this alone does *not* necessarily imply GDPR compliance as stored inferred data can still qualify as lacking compliance with the GDPR.

There are some important limitations of our analysis. Due to hardware and time constraints, we are only effectively able to model up to 8 users. Clearly, this is not a realistic full application simulation - i.e. a tool like Google Docs text prediction is predicated on millions of users. However, our project aims to demonstrate general, per-thread trends in stricter models of compliance.

In this paper, we explore the impact of GDPR compliance on federated machine learning. We discuss our work in building a tool to simulate user interaction with a federated machine learning model under different modes of GDPR compliance. In section 2, we discuss prior work related to the GDPR and federated machine learning. In section 3, we discuss the design of our tool to simulate user interaction with a federated machine learning model and the different modes of GDPR compliance. In section 4, we discuss more in-depth implementation details and our distribution of work. In section 5, we discuss results that came from experimentation with our tool.

2 Background/Related Work

Previous work on Federated Machine Learning [8] has come up with a system design with three major components: Devices, a central server, and a persistent storage [1]. In the design, the training happens locally on the devices, after which weight updates are going to be sent to the central server, which will perform an algorithm to combine the different weights together, and store the resulting model in the persistent storage. Additionally, `FederatedAveraging` has been proposed as an algorithm to combine users' weight contribution that is robust to unbalanced, non-IID data distributions in an efficient manner [7]. This has laid out a great groundwork for privacy-conscious Machine Learning, as a decentralized system where the only role of the server is to combine the weights and not store user data is a prerequisite to preventing server-side attack and the stealing of user data. However, the design proposed lacked the ability to effectively opt a user out of the training rounds that they have participated in, as all the information that are stored after each round is the global weights that result from it. The problem can be exacerbated as many users are not aware of the training happening on their phone (at night, when the phone is plugged in and usually when people are asleep) [1]. In addition to not being able to opt out of previous training rounds, the current design lacks specification on how users know which training rounds they participated in, and which data samples on their phone were implicated. This is a core motivation of our work, as we hope to design a system that allows users to opt their data out and know their participation in data inference activities, complying with the privacy laws like the GDPR.

Previous work has demonstrated a great range of tasks that the Federated Learning framework can apply to: Federated Matrix Factorization [3], neural network training and stochastic gradient descent jobs [7], or federated transfer learning [6]. As a proof of concept, we demonstrate that the framework we are proposing works with the specific tasks of learning to classify MNIST characters with a multi-layer perceptron algorithm, and to train a stochastic gradient descent model on a dataset that predicts whether an individual has an income over $\$50,000$. However, as the units of the training information (that engineers will feed into the system as they want to start a new training round) are defined in terms of functions (functions to select the training data on each participating device, to aggregate the data, or to update the local models based on the new global model), our system will generalize to aggregation functions other than `FederatedAveraging` and to tasks other than the two we are experimenting on.

Previous work has researched the methods to perform machine learning tasks over encrypted data ([2], [4], [5]). Although our work approaches privacy more from a system design level than through a encryption algorithm, our system would only work when combined with an encryption scheme on the server and persistent storage side. This is because as the persistent storage store crucial information about user participation (the rounds that they participated in and the associated weight contribution). Additionally, given the decentralized nature of the system, attacks to the system can be done through compromising the users' devices. This can be done through tech experts changing the code on users' devices to detect and record the communications between users and servers, or through network interceptors detecting communication between different agents in the system. Additionally, the system can be compromised if the persistent storage is compromised. Here, we recommend the methods proposed to enhance the robustness of decentralized contact tracing systems, as they share a similar structure of communication [10]. Notably, having the local applications on user devices run on a Trusted Platform Module can (1) verify the identity of users who are trying to access information through the application, (2) ensure that users cannot modify or access the code base to get unauthorized access to communication between users and the server.

3 Design

3.1 Compliance modes

Recall that GDPR compliance includes handling user requests to update or delete their data - including implied data. As such, we define strength of compliance across two dimensions:

1. The *frequency* with which modifications to user data are handled.

2. The *degree* to which data can be considered "implied"

By frequency, we mean that the server may decide to batch a large number of requests before handling them. Under "no" compliance, the server never handles user notifications of modified data. Under "weak," "strong," and "strict" compliance modes, the server handles user requests if the total number of user requests reaches 10000, 100, and 1 respectively. That is, under strict compliance, the server handles every user request as soon as reads that such a request has happened.

By degree, we mean that certain global model weights are more impacted by a particular user's contribution than others. For example, if a user provides local weights for a round r, we say that the global model is directly implied from that user's local data. However, if the next round r' does not call on that user to train but is dependent on the previous state of the global model, we say that the resulting global model is indirectly implied from that user's local data. We define GDPR compliance, in an abstract sense, to be that implied data should be handled by a user's request. That is, if the global model in r contains data that is implied from a user u, and then the server receives notice that u has deleted its data, that the global model should be updated as well. This definition is necessary because, although u might not have used the impacted data in training, the server is completely agnostic to user data and must assume that all user data was used during training.

Our design, hence, allows for four different modes of compliance:

1. **No compliance** In this setting, the model performs exactly like a generic Federated Machine Learning model.

2. **Weak compliance** In this setting, a `User` has the ability to request the `Aggregator` to delete/update their data that has participated in previous training rounds. The delete/update on the server side will be *shallow*, meaning only the participation information of the `User` i in the `Round` r will be deleted/updated (including their weight contribution, and the information about their participation in the round). If there is an update of the global weights after retraining `Round` j, this update will not cascade to the subsequent `Round`s $r+1, r+2, ...$

3. **Strong compliance** Unlike the *Weak compliance* case, when a `User` i requests an update on the `Aggregator` side with their data that participated in a `Round` r, the deletion/update will not only be for round j (which entails retraining the `Round`, and updating the information in the persistent storage `Logger`), but it will cascade to the subsequent `Round`s until the very end. With cascading updates, this operation is very computationally expensive. We address this by implementing a queue of requests, such that after a queue reaches a certain capacity, the `Aggregator` will request the updates to happen altogether, so the retraining operations can process all the requests from different Users at once.

4. **Strict compliance** This mode is stricter than the *Strong compliance* mode in that the requests coming from the `User`s would be handled right away. Deep deletions and updates, in this case, would be largely computationally expensive at the expense of immediate privacy.

3.2 Round

As mentioned earlier, in order to make our design extensible to a wide variety of tasks and aggregation function. The unit elements in a Round object are, therefore, functions. In order to create a `Round`, an engineer would need to define the following:

1. **Data function**: A function that takes in only data points that a user has opted in for training, and output a list of data points per the requirements from the engineer.

2. **Training function**: A function that is used by users on local devices to train on the data selected, and output the weight contribution that is going to be sent back to the server

3. **Aggregation function**: A function that is used by the Aggregator, after having received the weights from local devices, to combine the weight contributions and update the global model. This can be `FederatedAveraging` algorithm, or can be anything else.

A `Round` also carries a unique `Round ID`, and its information is going to be stored in the persistent storage (`Logger`) for later access, in case retraining needs to happen. The storage of `Round` and `Round ID` is necessary, as information about previous rounds needs to be accessed to perform the training in those rounds again, in the case a user requests all their participation in that `Round` to be deleted/updated.

3.3 User

Unlike previous work that has stopped short at mentioning that the user will only store their data on their device, we specify a few components on the user side that would make the design GDPR-compliant. Firstly, when the user adds a new piece of data in the storage, they will have a chance to specify whether the piece of data (e.g. an image, a row of excel sheet, etc.) is opted in for inference tasks by the central server or not. At any point, the user will have the ability to opt out their data from future training rounds by changing this `opt_in` variable associated with the data point to be `False`. When the Aggregator sends a training request to the User, the local participating data will have to go through two filtering rounds before it partakes in the training: only the data that is opted in for training can be candidates for the local data selection function sent by the Aggregator. After each training round, the User will locally associate the participating data points with the round information, so the User can access this information, potentially through a front-end application.

When a user deletes or updates their data, there will be an option for them to choose whether this update is local (e.g. you just delete a picture on your phone, without reflecting it on the server), or whether it should be propagated to update your participation in the previous training rounds that might have

concerned this data item. These deletion and update requests are going to be stored on a queue until the user explicitly asks the server to process all the requests in the queue.

3.4 Aggregator

In our design, the `Aggregator` remains an agent to communicate the training information with the Users, aggregate the resulting weight contributions, and communicate with the `Logger` to store metadata information that results from the `Round` (final global weights, which users participated, what was their contribution, etc.) As mentioned earlier, the `Aggregator` needs to accommodate a queue of delete and update requests from different `User`s, so after these queues get to a certain capacity, the deletion/updates will happen.

3.5 Logger

The `Logger`, in our design, stores the following information:

1. **User ID and Round ID mappings** to the actual `User`s and `Round`s. The logger needs a way of identifying `User`s (without finding out information about who they really are in real life), `Round`s, and `User`s' participation in different `Round`s. This is also for easy retrieval of appropriate `Round`s and `User`s for post-hoc training.

2. **Users' weight contribution in different Rounds** In a scenario where thousands (n) of `User`s participate in a `Round`, it would be extremely suboptimal to perform retraining locally on $n-1$ devices while only 1 `User` requested to delete/update their contribution. Therefore, there is a need to store the *weight contribution* of `User`s in a `Round`, such that we only need to perform local recomputation on `User`s who requested to delete/update their participation, and let the `Aggregator` combine the resulting new weight contributions with the old ones from other `User`s in the `Round`. This storage of weight contribution needs to be coupled with secure encryption schemes in order to prevent `User`'s weight contributions from being compromised, although in the worst case that it is, it would not be as bad as a central server with all the `User`'s raw data being compromised.

3. **Global models** Similar to the original Federated Learning system design [1], the persistent storage/`Logger` needs to store the global model. However, as there is an importance in our system to not just store the latest global model (so that we can effectively pinpoint the `Round` with which deletions/updates to the global model need to start and protect the global models in previous `Round`s), we are storing it as a mapping between `Round ID`s to the global models that result after the aggregation algorithm is performed.

3.6 Putting everything together

We built a tool to simulate different machine learning training techniques in a federated environment. An end users simply needs to run a python script with the command `python3 tester.py <num_tests> <num_users> <pct_gdpr_users> <pct_delete> <pct_update> <compliance_mode>`. This is done by having several different "users" maintain local data by adding and removing to and from it. A central "server" may, from time to time, request that users train on their local data. Users receive these requests and then send the results of that training back to the central server. In our implementation, we do not consider the corner case where a user might terminate their account or their application might fail in the middle of training. That is, we assume that user training will complete.

We did so by spawning several threads: one "server thread" and n "user threads" (where n is determined at the command-line). The server thread is responsible for requesting that particular users train in each training round[1]. Then, after receiving a training request, a "user thread" will perform the provided training function and send its results back to the central server. The server is then responsible for "aggregating" each of these results into a single global model.

At a lower level, the interaction between users and the server is done through a series of producer-consumer queues so as to ensure correct concurrency protocols. That is, when the server makes a request to the user to train, it places a tuple containing the current state of global weights and the training function into the queue, which the user will read from time to time. Similarly, after performing the training function provided by the server, the user places the results of the training and the associated loss into a different producer-consumer queue that the server reads from time to time. To be clear, this part of the implementation entails $2n$ producer-consumer queues. Each user is assigned two queues, one where it is consumer (to receive training requests) and one where it is producer (to send training results to the server). Similarly, the server is the producer of n queues (to send training requests) and the consumer of n queues (to receive results).

To implement GDPR compliance, users are given a third queue where they are the producer and the server is the consumer. In this queue, they provide a "note" that certain data may no longer be valid for training. The server then determines whether or not this will be handled, depending on how strictly it defines GDPR compliance.

In order to implement GDPR compliance, it must also be the case that the server must track metadata about which users it requested to train in which rounds, as well as previous weights and which users contributed to which weights. As a result, although the server is not tasked with storing local data, the server is responsible for storage that we maintain

[1]The number of rounds can be arbitrarily determined by the application, so in our implementation we perform n rounds of training.

in a "Logger" object. The server alone is responsible for maintaining this logger. Note that we implement a logger, even under "no" compliance, as it might be the case that this metadata is important for other server tasks. That is, it is not unreasonable that the server would maintain this logger even under "no" compliance.

We have also implemented a front-end web application with one of the two machine learning applications that we tested our design on. The web application is designed to demonstrate (1) general statistics of the training rounds, (2) the capabilities that different Users will have with different modes of compliance, and (3) their interaction with the Server. In order to get the full experience of the communication between the `Aggregator` and `Users`, we recommend running the testers, as described above. To run the web app, you can just simply do `python3 app.py` and follow the instructions.

4 Implementation

4.1 General

Our project is made up of about 1300 lines of code. There are three main modules that drive our system: "User," "Aggregator," and "Logger." The User module defines user capabilities, like maintaining a user's local data, and is driven by the user thread. That is, there is a User object per user thread and each User object is only accessible by that user thread. The Aggregator module defines the functionality of the central server, namely calling on users to train, handling user requests, etc.... Similarly, the Aggregator is driven by the server thread and is only accessible by the server thread. The Logger module maintains all global metadata about training rounds, and is only accessible by the server thread.

The training model functions are defined external to these modules, but are maintained in the server thread to send to user threads. This design decision makes applying different training functions trivial, as it merely requires a change in function call.

4.2 Use cases

We tested our design with two different Machine Learning tasks and algorithms.

1. **Stochastic Gradient Descent training on Income data**

   In the most basic use case, we tested to see if our system design works with a training scheme where the global models are in the form of weight arrays. The training happened on the Census Income dataset. The information about the dataset can be found here: https://archive.ics.uci.edu/ml/datasets/Census+Income. For the aggregation function, we followed the `FederatedAveraging` algorithm proposed in previous work [7]. This algorithm was demonstrated to work on Stochastic Gradient Descent type tasks, even when the data distribution was non-IID. The Stochastic Gradient Descent algorithm will be performed on each device.

2. **Multilayer Perceptron training on MNIST data**

   To test whether the system design works with a learning algorithm where the global models are neural networks and not just a weight matrix, we performed training on the MNIST dataset (more information here: http://yann.lecun.com/exdb/mnist/) with the global models being a Multilayer Perceptron with 4 different layers. We used *Cross Entropy Loss* as part of our training. The back-propagation updates to the neural network would happen locally, and the local devices will send the resulting neural network information to the `Aggregator`, which will perform the `FederatedAveraging` algorithm to come up with an updated global model.

4.3 Work distribution

As far as the distribution of work was concerned, Nam was largely responsible for reviewing the literature, implementing the framework for the `User`, `Aggregator`, and `Logger` classes, and implementing the two use cases to test our design upon. Sam was largely responsible for the integration of each of these components as well as setting up the system that runs these components. Both Sam and Nam participated in debugging the system, and both Sam and Nam were responsible for designing the implementation of the system. The Github commit log reflects these remarks. Our code and data can be found here: https://github.com/samueltphd/NLP-GDPR.

5 Evaluation

We begin our discussion of evaluation by discussing the assumptions made in our system. We do not add any latency to adding or removing items to a producer-consumer queues, thereby assuming that users and the server can communicate with infinite bandwidth. Furthermore, as previously discussed, we assume that the central server would maintain a Logger for tracking metadata about training, even though the Logger is not utilized in the "no" compliance mode. Finally, we do not add any additional tasks to the server or users. That is, their only responsibility is to train and maintain data.

Given our assumptions, we perform two primary benchmarks, each with different variations. The first benchmark assumes that all users are aware of the GDPR, and that 50% of their behaviors are to remove their existing data and 50% of their behaviors are to update the value of their existing data[2].

[2]Note that the server does not necessarily need to handle updates to data in order to remain GDPR compliant, but this may help accuracy metrics over time. For the sake of our project, the server only handles deletions.

compliance	thds	time (sec)	size (MB)
no, 10-10	1	83.094	2019245.6
no, 50-50	1	77.589	1658294.4
weak, 10-10	1	83.388	2023348
weak, 50-50	1	78.064	1659318.4
strong, 10-10	1	82.594	2024472
strong, 50-50	1	78.683	1658576
strict, 10-10	1	83.301	2023185.6
strict, 50-50	1	78.192	1658153.6
no, 10-10	8	1038.176	15606279.2
no, 50-50	8	409.122	1648156.8
weak, 10-10	8	1063.853	15316912.8
weak, 50-50*	8	599.668	1647484
strong, 10-10	8	1046.714	15725040.8
strong, 50-50*	8	533.625	1648550.4
strict, 10-10	8	1168.074	15599537.6
strict, 50-50*	8	567.639	1649940.8
no, 10-10	32	55277.527	270822992.8

Table 1: Results of training benchmarks. Note: * refers to an error in training that caused the server thread to crash, so results for 50-50 benchmark cannot be compared to 10-10 benchmark.

The second benchmark also assumes that all users are aware of the GDPR, and that 10% of their behaviors are to remove their existing data, 10% of their behaviors are to update the value of existing data, and 80% of their behaviors are to add new data to their local data.

In each of these benchmarks, we ran experiments with 1 user, 8 users, and 32 users. Furthermore, we ran analyses with a simple (non-neural network based) approach to a income binary classification problem and a more complicated approach to the MNIST classification (10 classes) problem. We measured time and space used in each of these tests. We ran five tests under each configuration. Unfortunately, due to limitations in time and available computational resources, we were only able to obtain results to the income tests and only up to 8 threads[3].

Fig. 1 shows the number of requests handled in different thread counts. That is, under "no" compliance, no user-provided update requests are handled by the server thread with either 1 user or 8 users - as we expect. Under "weak" compliance, no update requests are handled with 1 user and 9 requests (out of a possible 40) are handled with 8 users. This is because, with one user, the server never receives enough requests to trigger handling them. Similarly, under "strong" compliance, no requests are handled with 1 user, but 37 requests are handled with 8 users. Under "strict" compliance, 5 (out of a possible 5) requests that handled and, with 8 users threads, 40 requests are handled. This confirms that our compliance models are handling *frequency* as we expect.

[3]This is evident when noting the time scale of even the more simple tests with 8 users takes about an hour per test to run.

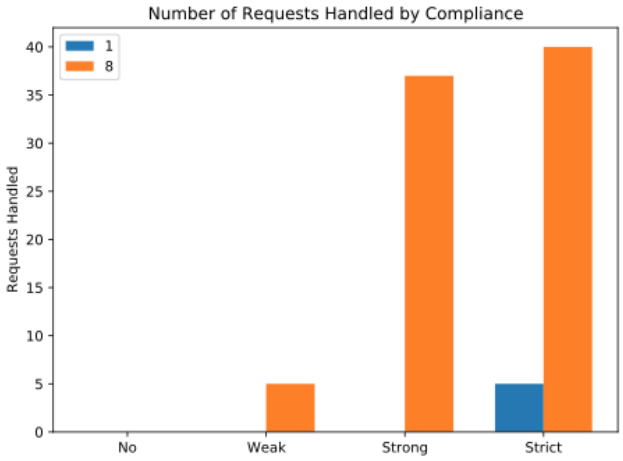

Figure 1: Number of requests handled by thread count and by compliance level.

Although incomplete, there are still several takeaways to be made from table 1. Note that we use "no" GDPR compliance as a control group. That is, our results demonstrate that we expect about a 1.5x increase in time when comparing 1 thread and 8 threads and about a 16x increase in time per thread when comparing 8 threads and 32 threads. However, we note that we ran our experiments on a four-core machine, so the increase in time per thread could very well be a result of overusing system capacity. It would be interesting to run similar experiments on a system with more cores to see if such time increase is still prevalent. Similarly, our control group demonstrates that there is 196% increase per thread in space required when comparing 1 thread to 8 threads and a 155% increase in space from 8 threads to 32 threads.

When looking at the time increases with "strict" compliance, however, we see that there is a 1.75x increase in time as compared to a 1.5x increase in time with "no" compliance. When considering space, there is both a 196% increase under "no" compliance and "strict" compliance. That is, there is no additional space penalty for implementing even a very strict definition of GDPR compliance.

6 Conclusion

In our project, we explored the impact of GDPR compliance on federated machine learning models for different tasks. In doing so, we introduce two different definitions of GDPR compliance, namely *frequency* and *degree*, and explore different permutations of their integration. From there, we determine that the strictest definitions of GDPR compliance can introduce minor time overheads (of about 1.75x increase in time per user as compared to a baseline of 1.5x increase in time) while no significant additional space overhead is required.

7 Acknowledgements

We would like to say thank you to Professor Malte Schwarzkopf for an amazing semester with Brown *CS2390*, and for all the invaluable feedback and support that he has been giving us on this project and beyond throughout the semester. Nam also would like to say thank you to Amy Pu '21 who helped us with debugging our multilayer perceptron, and who has been supporting Nam personally through this *interesting* semester. Nam additionally would like to say thank you to Bashar Zaidat '21 and Rajyashri Reddy '21 for helping Nam double checking his Stochastic Gradient Descent algorithm implementation. Last but not least, while building the system, we referenced ours against design by Shaoxiong Ji (link here: https://github.com/shaoxiongji/federated-learning), which we would like to thank him for.

References

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

[2] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.

[3] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. Secure federated matrix factorization. *IEEE Intelligent Systems*, 2020.

[4] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[5] Angela Jäschke and Frederik Armknecht. Unsupervised machine learning on encrypted data. In *International Conference on Selected Areas in Cryptography*, pages 453–478. Springer, 2018.

[6] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 2020.

[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[8] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. *Google Research Blog*, 3, 2017.

[9] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, et al. Decentralized privacy-preserving proximity tracing. *arXiv preprint arXiv:2005.12273*, 2020.

[10] Serge Vaudenay. Analysis of dp3t between scylla and charybdis.(2020), 2020.