DOI:10.1145/3500240

Article development led by acmqueue
queue.acm.org

Building privacy-preserving systems for machine learning and data science on decentralized data.

BY KALLISTA BONAWITZ, PETER KAIROUZ, BRENDAN MCMAHAN, AND DANIEL RAMAGE

Federated Learning and Privacy

MACHINE LEARNING AND data science are key tools in science, public policy, and the design of products and services thanks to the increasing affordability of collecting, storing, and processing large quantities of data. But centralized collection can expose individuals to privacy risks and organizations to legal risks if data is not properly managed. Starting with early work in 2016,[13,15] an expanding community of researchers has explored how data ownership and provenance can be made first-class concepts in systems for learning and analytics in areas now known as *federated learning* (FL) and *federated analytics* (FA).

With this expanding community, interest has broadened from the initial work on federations of mobile devices to include FL across organizational silos, Internet of Things (IoT) devices, and more. In light of this, Kairouz et al.[10] proposed a broader definition:

Federated learning *is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective.*

An approach very similar in both philosophy and implementation, federated analytics[17] can be taken to allow data scientists to generate analytical insight from the combined information in decentralized datasets. While the focus here is on FL, much of the discussion on technology and privacy applies equally well to FA use cases.

This article provides a brief introduction to key concepts in federated learning and analytics with an emphasis on how privacy technologies may be combined in real-world systems and how their use charts a path toward societal benefit from aggregate statistics in new domains and with minimized risk to individuals and to the organizations who are custodians of the data.

Privacy Principles for Learning and Analytics

To ground a more detailed discussion of FL, let's begin by clarifying the relevant notions of privacy. Privacy is an inherently multifaceted concept, even when restricted to the realm of the products and services offered by a technology company, which is the focus here. Three key components of privacy are highlighted in this context: transparency and consent; data minimization; and anonymization of released aggregates.

Transparency and consent are foundational to privacy: they are how users of the product/service both understand and approve of the ways in which their data will be used. Privacy technology cannot replace transparency and consent, but data-stewardship approaches based on strong privacy technologies make it easier for all parties involved to reason about which

types of data usage might be possible (and which are ruled out by design), thereby enabling clearer privacy statements that are simpler to understand, verify, and enforce.

The role of privacy technology becomes clearer when considering specific goals that can be advanced by computation on privacy-sensitive user data; for example, improving a mobile keyboard's suggestions based on user input to the virtual keyboard. How can the keyboard be improved in as minimally invasive a manner as possible?

The computation goals are primarily the training of machine learning (ML) models (FL) and the calculation of metrics or other aggregate statistics on user data (FA). As we will see, both analytics and machine learning can be accomplished via appropriately chosen aggregations over (possibly preprocessed) user data. In this context, specializations of two broad privacy principles apply:

The principle of data minimization, as applied to aggregations, includes the objective to collect only the data needed for the specific computation (focused collection), to limit access to data at all stages, to process individuals' data as early as possible (early aggregation), and to discard both collected and processed data as soon as possible (minimal retention). That is, data minimization implies restricting access to all data to the smallest set of people possible, often accomplished via security mechanisms, such as encryption at rest and on the wire, access-control lists, and more nascent technologies such as secure multiparty computation and trusted execution environments, to be discussed later.

The principle of data anonymization captures the objective that the final released output of the computation does not reveal anything unique to an individual. When this principle is specialized to *anonymous aggregation*, the goal is that data contributed by any individual user to the computation has only a small (limited, measured, and/or mitigated) influence on the final aggregate output. For example, aggregate statistics, including model parameters, when released to an engineer—or beyond—should not vary significantly based on whether any particular user's data was included in the aggregation. The XKCD comic shown here illustrates a humorous example where this principle is not respected, but this

IMAGE BY VLADGRIN

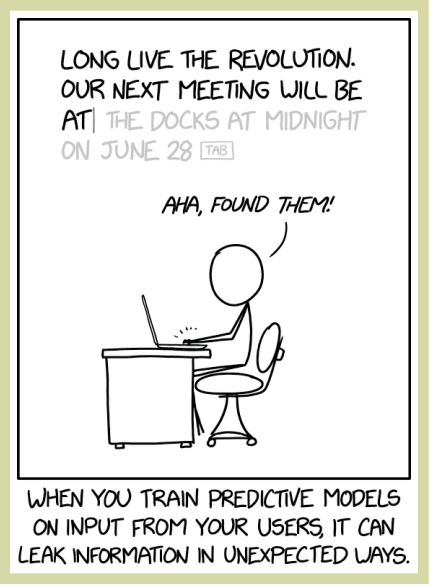
memorization phenomenon has been shown to be a real issue for modern deep networks.[7,8]

Another way to view these principles is that data minimization pertains to *how* the computation is executed and data is handled, while data anonymization pertains to *what* is computed and released.

By design, FL structurally embodies data minimization. Figure 1 compares the federated approach to more standard centralized techniques. Critically, data collection and aggregation are inseparable in the federated approach—purpose-specific transformations of client data are collected for immediate aggregation, with analysts having no access to per-client messages. FL and FA are instances of a general federated computation schema that embodies data-minimization practices. The more typical approach of centralized processing replaces on-device preprocessing and aggregation with data collection, with the primary minimization happening on the server during the processing of the logged data.

The ML and analytics goals considered here are compatible with the objective of anonymous aggregation. With ML, the goal is to train a model that predicts accurately for all users, without overfitting (memorizing) the data used for training. Similarly, with statistical queries the goal is to estimate population statistics, which should again not be too significantly influenced by any one user's data.

FL can be combined with other techniques (particularly differential privacy and privacy/memorization auditing, treated in more depth later) to ensure released aggregates are sufficiently anonymous. This situation contrasts the privacy relationship you might have with a bank or healthcare provider, where the data anonymization principle may not apply since direct access by the provider to an individual's sensitive data cannot be avoided; in these interactions, trust in the provider to use the data only for the intended purpose is the fundamental tenet.



Federated Learning Settings and Applications

As indicated earlier, the defining characteristics of FL include keeping raw data decentralized and learning via aggregation. This assumption of locally generated data—often heterogeneous in distribution and quantity—distinguishes FL from more typical datacenter-based distributed learning settings, where data can be arbitrarily distributed and shuffled, and any worker node in the computation can access any of the data.

The role of a central orchestrator is practically useful and often necessary, as in the case of mobile devices that lack fixed IP addresses and require a central server to mediate device-to-device communication. It further constrains the space of relevant algorithms and helps to distinguish FL from more general forms of decentralized learning, including peer-to-peer approaches.

From the basic definition, two FL settings have received particular attention:

- Cross-device FL, where the clients are large numbers of mobile or IoT devices.
- Cross-silo FL, where the clients are a typically smaller number of organizations, institutions, or other data silos.

The accompanying table, adapted from Kairouz et al.,[10] summarizes the key characteristics of the FL settings and highlights some of the key differences between the cross-device and cross-silo settings, as well as contrasting with datacenter distributed learning.

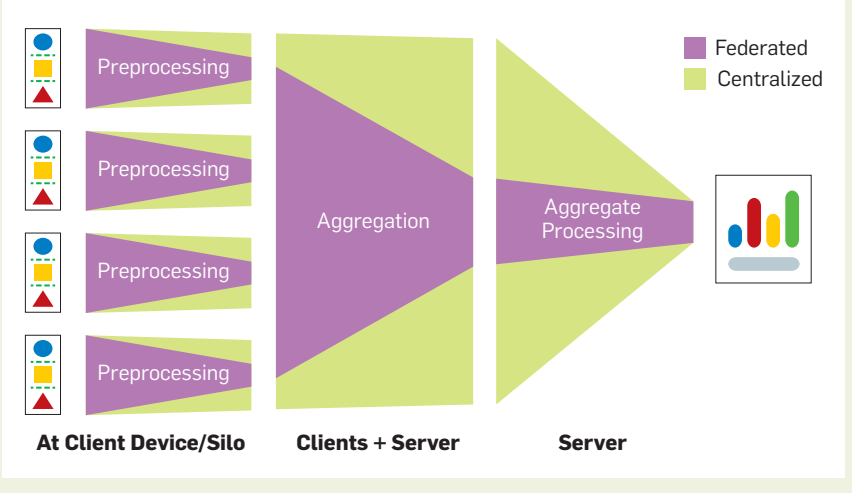
Cross-device FL is now used by both Google[6] and Apple[16] for Android and iOS phones, respectively, for many applications such as mobile keyboard prediction; cross-silo FA is being explored for problems such as health research (for example, Google Health Studies[a]).

Cross-silo FL has received considerable attention as well. Health and medical applications are a primary motivation, with significant investments from Nvidia, IBM, and Intel, as well as numerous startups. Another application that is on the rise is finance, with investments from WeBank, Credit Suisse, Intel, and others.

Algorithms for Cross-Device Federated Learning

Modern ML approaches, particularly deep learning, are generally data hungry and compute-intensive, and so the fea-

Figure 1. Data minimization in federated vs. centralized approaches.



a https://blog.google/technology/health/google-health-studies-app/

Typical FL settings and of traditional distributed learning.

	Datacenter Distributed Learning	Cross-Silo Federated Learning	Cross-Device Federated Learning
Setting	Training a model on a large but "flat" dataset. Clients are compute nodes in a single cluster or datacenter.	Training a model on siloed data. Clients are different organizations (for example, medical or financial) or datacenters in different geographical regions.	The clients are a very large number of mobile or IoT devices.
Data distribution	Data is centrally stored, so it can be shuffled and balanced across clients. Any client can read any part of the dataset.	Data is generated locally and remains decentralized. Each client stores its own data and cannot read the data of other clients. Data is not independently or identically distributed.	
Orchestration	Centrally orchestrated.	A central orchestration server/service organizes the training, but never sees raw data.	
Distribution scale	Typically 1—1,000 clients.	Typically 2—100 clients.	Up to 10^{10} clients.
Client properties	Clients are reliable and almost always available to participate in computations. Clients may be directly addressed, and can maintain state across computation rounds.		Clients are often unavailable and can only be accessed by random sampling from available devices. For large populations a single client will typically only participate once in a given computation.

sibility of the federated training of production-quality models was far from a foregone conclusion. Much of our early work, particularly the 2017 paper, "Communication-efficient Learning of Deep Networks from Decentralized Data,"[13] focused on establishing a proof of concept. This work introduced the federated averaging algorithm, which continues to see widespread use, though many variations and improvements have since been proposed.

The core idea builds on the classic stochastic gradient descent (SGD) algorithm, which is widely used for the training of ML models in more traditional settings. The model is given as a function from training examples to predictions, parameterized by a vector of model weights, and a loss function that measures the error between the prediction and the true output (label). SGD proceeds by sampling a batch of training examples (typically from tens to thousands), computing the average gradient of the loss function with respect to the model weights, and then adjusting the model weights in the opposite direction of the gradient. By appropriately tuning the size of the steps taken on each iteration, SGD can be shown to have desirable convergence properties, even for nonconvex functions.

The simplest extension of SGD to the federated setting would be to broadcast the current model weights to a random set of clients, have them each compute the gradient of the loss on their local data, average these gradients across clients at the server, and then update the global model weights. SGD, however, often requires 10^5 or more iterations to produce a high-accuracy model. Back-of-the-envelope calculations suggest a single iteration might take minutes in the federated setting, implying federated training might take between a month and a year—outside the realm of practicality.

The key idea of federated averaging is intuitive: Decrease communication and startup costs by taking multiple steps of SGD locally on each device, and then average the resulting models (or model updates) less frequently. If models are averaged after each local step, this reduces to SGD (and is probably too slow); if models are averaged too infrequently, they might diverge, and averaging could produce a worse model. Is there a sweet spot in between? Empirically, the 2017 paper[13] showed the answer is yes, demonstrating that moderate-sized language models (for example, for next-word prediction) and image-classification models could be trained in fewer than 1,000 communication rounds. This reduces the expected training time to a few days—still much slower than would be possible with a high-performance compute cluster on centralized data, but within the realm of feasibility for real-world production use.

This algorithm also demonstrates the key privacy point mentioned earlier—that model training can be reduced to the (repeated) application of a federated aggregation (the averaging of model gradients or updates), as in Figure 1.

Workflows and Systems for Cross-Device Federated Learning

Having a feasible algorithm for FL is a necessary starting point but making cross-device FL a productive approach for ML-driven product teams requires much more. Based on Google's experience deploying cross-device FL across multiple Google products, the typical workflow often includes the following steps:

1. Identifying a problem well-suited for FL. Typically this means a moderately sized (1MB–50MB) on-device model is desired; training data potentially available on-device is richer or more representative than data available in the datacenter; there are privacy or other reasons to prefer not to centralize the data; and the feedback signals (labels) necessary to train the model are readily available on-device (for example, a model for next-word prediction can naturally be trained based on what users type if they ignore predicted next words; an image-classification model would more difficult to train unless interaction with the app naturally led to labeled images).

2. Model development and evaluation. As with any ML task, choosing the right model architecture and hyperparameters (learning rates, batch sizes, regularization) is critical to success in FL. The challenge can be bigger in the federated setting, which introduces a number of new hyperparameters (for example, number of clients participating in each round, how many local steps to take before averaging). Often the starting point is to do coarse model selection and tuning using a *simulation*

of FL based on proxy data available in the datacenter. Final tuning and evaluation must be conducted using federated training on real devices, however, as the differences in data distribution, real-world device fleet characteristics, and many other factors are impossible to capture fully in simulation. Evaluation must also be conducted in a federated manner: Independent from the training process, the candidate global model is sent to (held-out) devices so that accuracy metrics can be computed on these devices' local datasets and aggregated by the server (both simple averages and histograms over per-client performance are important). Taken together, these needs give rise to two key infrastructure requirements: providing high-performance FL simulation infrastructure that allows a smooth transition to running on real devices; and a cross-device infrastructure that makes it easy to manage multiple simultaneous training and evaluation tasks.

3. Deployment. Once a high-quality candidate model is selected in step 2, the deployment of that model (for example, making user-visible next-word predictions in a mobile keyboard) typically follows the same procedures that are used for a datacenter-trained model: additional validation and testing (potentially including manual quality assurance), live A/B testing to compare to the previous production model, and a staged rollout to the full device fleet (potentially several orders of magnitude more devices than actually participated in the training of the model).

It is worth emphasizing that all the work in step 2 has no impact on the user experience of the devices participating in training and evaluation; models being trained with FL don't make predictions visible to the user unless they go through the deployment step. Ensuring that this processing doesn't otherwise negatively impact the device is a key infrastructure challenge. For example, heavyweight computation might execute only when the devices are idle, plugged in, and on an unmetered Wi-Fi network.

Figure 2 illustrates the model development and deployment workflows. Building a scalable infrastructure and compelling developer APIs for these workflows is a significant challenge. A paper by Bonawitz et al.[6] provides an overview of Google's production system as of 2019.

Privacy for Federated Computations

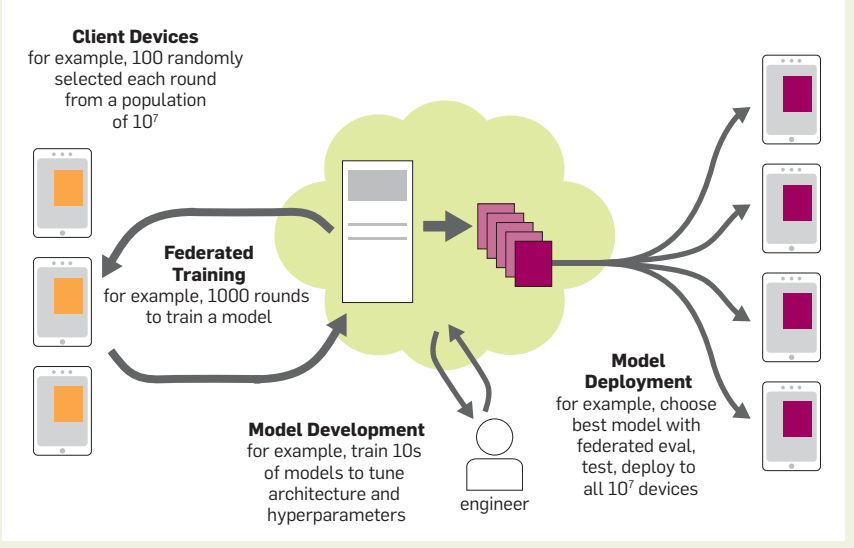
FL provides a variety of privacy advantages out of the box. In the spirit of data minimization, the raw data stays on the device, and updates sent to the server are focused on a particular purpose, ephemeral, and aggregated as soon as possible. No non-aggregated data is persisted on the server, end-to-end encryption protects data in transit, and both the decryption keys and decrypted values are held only ephemerally in RAM. ML engineers and analysts interacting with the system can access only aggregated data. The fundamental role of aggregates in the federated approach makes it natural to limit the influence of any individual client on the output, but algorithms need to be carefully designed if the goal is to provide more formal guarantees such as differential privacy.

Researchers at Google and beyond are strengthening the privacy guarantees that an FL system can make. While the basic FL approach has proven feasible and gained substantial adoption, its combination with other techniques described in this section is still far from "on by default for most uses of FL." Even as the state-of-the-art advances, inherent tensions with other objectives (including fairness, accuracy, development velocity, and computational cost) will likely prevent a one-size-fits-all approach to data minimization and anonymization. Thus, practitioners benefit from continued advancement of research ideas and software implementations for composable privacy enhancing techniques. Ultimately, decisions about privacy technology deployment are made by product or service teams in consultation with domain-specific privacy, policy, and legal experts. Our obligation as privacy technologists is twofold: to enable products to offer more privacy through usable FL systems and, perhaps more importantly, to help policy experts strengthen privacy definitions and requirements over time.

In analyzing the privacy properties of a federated system, it is useful to consider access points and threat models. Building on Figure 2, one can ask what private information an actor might learn with access to various parts of the system. With access to the physical device or network? With root or physical access to the servers providing the FL service? To the models and metrics released to the ML engineer? To the final deployed model?

The number of potentially malicious parties varies dramatically as information flows through this system. A very small number of parties should have physical or root access to the coordinating server, for example, but nearly anyone might be able to access the final model shipped out to a large fleet of smartphones.

Figure 2. Components and phases of a cross-device FL system.



Privacy claims must therefore be assessed for a complete end-to-end system. A guarantee the final deployed model has not memorized user data may not matter if suitable security precautions are not taken to protect the raw data on the device or an intermediate computation state in transit. Other techniques can provide even stronger guarantees.

Figure 3 shows threat models for an end-to-end FL system and the role of data minimization and anonymous aggregation. Data minimization addresses potential threats to the device, network, and server by, for example, improving security and minimizing the retention of data and intermediate results. When models and metrics are released to the model engineer or deployed to production, anonymous aggregation protects individuals' data from parties with access to these released outputs.

Data Minimization for Aggregation

At several points in a federated computation, the participants expect one another to take the appropriate actions, and only those actions. For example, the server expects the clients to execute their preprocessing step accurately; the clients expect the server to keep their individual updates a secret until they have been aggregated; both the clients and the server expect that neither the data analyst nor the deployed ML model user will be able to extract an individual's data; and so on.

Privacy-preserving technologies support the structural enforcement of these interparty expectations, preventing participants from deviating even if they happen to be malicious or compromised. In fact, FL systems can be viewed as a kind of privacy-preserving technology in themselves, structurally preventing the server from accessing anything about a client's data that was not included in the update submitted by that client.

Take, for example, the aggregation phase of FL. An idealized system might imagine a completely trusted third party who aggregates the clients' updates and reveals only the final aggregate to the server. In reality, no such mutually trusted third party typically exists to play this role, but various technologies allow an FL system to *simulate* such a third party under a wide range of conditions.

For example, a server could run the aggregation procedure within a *secure enclave*—a specially constructed piece of hardware that can not only prove to the clients what code it is running, but also ensure no one (not even the hardware's owner) can observe or tamper with the execution of that code. Currently, however, the availability of secure enclaves is limited, both in the cloud and on consumer devices, and available enclaves may implement only some of the desired enclave properties (secure measurement, confidentiality, and integrity[19]). Moreover, even when available and full-featured, secure enclaves may come with additional limitations, including very limited memory or speed; vulnerability to data exposure via side channels (for example, cache-timing attacks); difficult-to-verify correctness (because of proprietary implementation details); dependence on manufacturer-provided attestation services (and key secrecy); and so on.

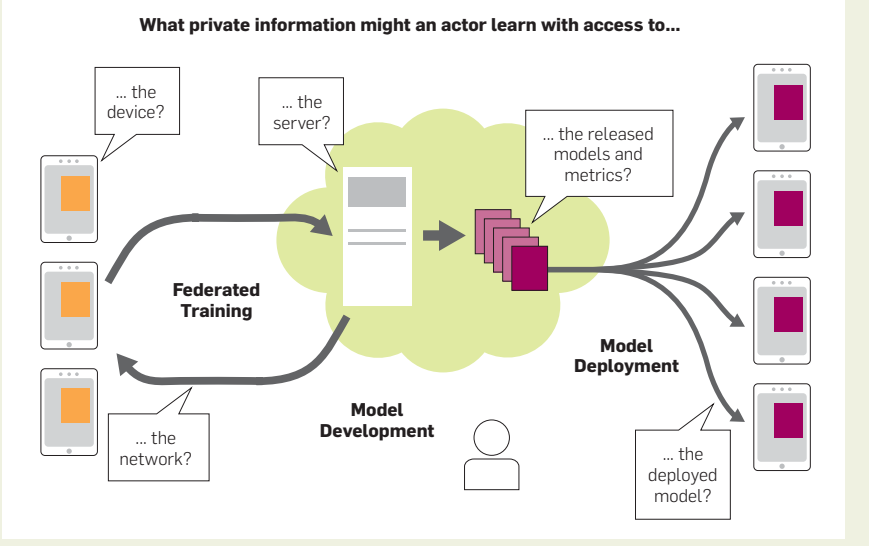
Distributed cryptographic protocols for secure multiparty computation can be used collaboratively to simulate a trusted third party without the need for specialized hardware, so long as a sufficiently large number of the participants behave honestly. While secure multiparty computation for arbitrary functions remains computationally prohibitive in most cases, specialized secure aggregation algorithms for vector summation in the federated setting have been developed that provably preserve privacy even against an adversary that observes the server and controls a significant fraction of the clients, while maintaining robustness against clients dropping out of the computation.[5] Such algorithms are both:

- *Communication efficient* - $O(\log n + \ell)$ communication per client, where n is the number of users and ℓ is the vector length, with small constants yielding less than twice the communication of aggregation in the clear for a wide range of practical settings; and
- *Computation efficient* - $O(\log^2 n + \ell \log n)$ computation per client.[3]

Cryptographic secure aggregation protocols have been deployed in commercial federated computing systems for years.[6,17]

Beyond private aggregation, privacy-preserving technologies can be used to secure other parts of an FL system. For example, either secure enclaves or cryptographic techniques (for example, zero-knowledge proofs) can ensure that the server can trust that clients have preprocessed faithfully. Even the model broadcast stage can benefit: For many learning tasks, an individual client may have data relevant to only a small portion of the model; in this case, the client can privately retrieve just that segment of the model for training, again using either secure enclaves or cryptographic techniques (for example, private information retrieval) to

Figure 3. Threat models for a FL system.



ensure the server learns nothing about the segment of the model for which the client has relevant training data.

Computing and Verifying Anonymous Aggregates

While secure enclaves and private aggregation techniques can strengthen data minimization, they are not designed specifically to produce anonymous aggregates—for example, limiting the influence of a user on the model being trained. Indeed, a growing body of research suggests that the learned model can (in some cases) leak sensitive information.[8]

The gold-standard approach to data anonymization is differential privacy (DP).[9] For a generic procedure that aggregates *records* in a *database*, DP requires bounding any record's contribution to the aggregate and then adding an appropriately scaled random perturbation. For example, in DP-SGD (differentially private stochastic gradient descent) you clip the ℓ_2 norm of the gradients, aggregate the clipped gradients, and add Gaussian noise in each training round.[1]

Differentially private algorithms are necessarily randomized, and hence you can consider the *distribution* of models produced by an algorithm on a particular dataset. Intuitively, differential privacy says this distribution over models is similar when the algorithm is run on input datasets that differ by a single record. Formally, DP is quantified by privacy loss parameters (ε, δ), where a smaller (ε, δ) pair corresponds to increased privacy. A randomized algorithm A is (ε, δ)-differentially private if for all possible outputs (for example, models) m, and for all datasets D and D' that differ in, at most, one record:

$$P(A(D) = m) \leq e^{\varepsilon} P(A(D') = m) + \delta.$$

This goes beyond simply bounding the sensitivity of the model to each record by adding noise proportional to any record's influence, therefore ensuring sufficient randomness to mask any one record's contribution to the output.

In the context of cross-device FL, a *record* is defined as *all* the training examples of a single user/client.[14] This notion of DP is referred to as user-level DP and is stronger than example-level DP, where a record corresponds to a single training example, because in general one user may contribute many training examples. Even in centralized settings, FL algorithms are well suited for training with user-level DP guarantees, because they compute a single update to the model from all of a user's data, making it much easier to bound each user's total influence on the model update (and hence on the final model).

Providing formal (ε, δ) guarantees in the context of cross-device FL systems can be particularly challenging because the set of all eligible users is dynamic and not known in advance, and the participating users may drop out at any point in the protocol. While the recent work of Balle et al.[2] suggests these challenges can be overcome in theory, building an end-to-end protocol that works in production FL systems is still an important problem to solve.

In the context of cross-silo FL, the unit of privacy can take on a different meaning. For example, it is possible to define a record as all the examples on a data silo if the participating institutions want to ensure an adversary who has access to the model iterations or to the final model cannot determine whether or not a particular institution's dataset was used in the training of that model. User-level DP can still be meaningful in cross-silo settings where each silo holds data for multiple users. Enforcing user-level privacy, however, may be more challenging if multiple institutions have records from the same user.

Over the past decade, an extensive set of techniques has been developed for differentially private data analysis, particularly for the central or trusted-aggregator setting, where the raw (or minimized) data is collected by a trusted service provider that implements the DP algorithm. More recently, there has been great interest in the local model of DP,[12] where the data is perturbed on the client side before it is collected by a service provider. Local DP avoids the need for a fully trusted aggregator, but it is now well established that local DP leads to a steep hit in accuracy.

To recover the utility of central DP without having to rely on a fully trusted central server, an emerging set of approaches, often referred to as distributed DP, can be used.[4,11] The goal is to render the output differentially private before it becomes visible (in plaintext) to the server. Under distributed DP, clients first compute minimal application-specific reports, perturb these slightly with random noise, and then execute a private aggregation protocol. The server then has access only to the output of the private aggregation protocol. The noise added by individual clients is typically insufficient for a meaningful local DP guarantee on its own. After private aggregation, however, the output of the private aggregation protocol provides a stronger DP guarantee based on the total sum of noise added across all clients. This applies even to someone with access to the server under the security assumptions necessary for the private aggregation protocol.

For an algorithm to provide a formal user-level DP guarantee, it must not only bound the sensitivity of the model to each user's data, but also add noise proportional to that sensitivity. While the addition of sufficient random noise is required to ensure a small enough ε for the DP definition itself to offer a strong guarantee, empirically it has been observed that limiting sensitivity even with small amounts of noise (or no noise at all) can significantly reduce memorization.[18] This gap is to be expected, as DP assumes a "worst-case adversary'' with infinite computation and access to arbitrary side information. These assumptions are often unrealistic in practice. Thus, there are substantial advantages to training using a DP algorithm that limits each user's influence, even if the explicit random noise introduced into the training process is not enough to ensure a small ε formally. Nevertheless, designing practical FL and FA algorithms that achieve small ε guarantees is an important area of ongoing research.

Model auditing techniques can be used to further quantify the advantages of training with DP.[7,8,18] These techniques are empirical in nature and can be applied during or after training. They broadly include techniques that quantify how much a model overlearns (or memorizes) unique or rare training examples, and techniques that quantify to what extent it is possible to infer whether or not a user's examples were used during training. These auditing techniques are useful even when a large ε is used, as they can

quantify the gap between DP's worst-case adversaries and realistic ones with limited computational power and side information. They can also serve as a complementary technology for pressure-testing DP implementations: unlike the formal mathematical statements of DP, these auditing techniques are applied to complete end-to-end systems, potentially catching software bugs or mis-chosen parameters.

Federated Analytics

The focus of this article so far has primarily been on FL. Beyond learning ML models, data analysts are often interested in applying data science methods to the analysis of raw data that is stored locally on users' devices. For example, analysts may be interested in learning aggregate model metrics, popular trends and activities, or geospatial location heatmaps. All of this can be done using FA.[17] Similar to FL, FA works by running local computations over each device's data and making only the aggregated results available to product engineers. Unlike FL, however, FA aims to support basic data science needs, such as counts, averages, histograms, quantiles, and other SQL-like queries.

Consider an application where an analyst wants to use FA to learn the 10 most frequently played songs in a music library shared by many users. The same federated and privacy techniques described previously can be used to perform this task. For example, clients can encode which songs they have listened to into a binary vector of length equal to the size of the library and use distributed DP to ensure that the server sees only a differentially private sum of these vectors, giving a DP histogram of how many users have played each song. As this example illustrates, however, FA tasks can differ from FL ones in several ways:

1. FA algorithms are often noninteractive and involve rounds with a large number of clients. In other words, unlike FL applications, there are no diminishing returns from having more clients in a round. Therefore, applying DP is less challenging in FA since each round can contain a large number of clients, and fewer rounds are needed.

2. There is no need for the same clients to participate again in later rounds. In fact, clients that participate again may bias the results of the algorithm. Therefore, an FA task is best served by an infrastructure that limits the number of times any individual can participate.

3. FA tasks are typically sparse, making efficient private sparse aggregation a particularly important topic; many open research questions exist in this space.

It is worth noting that while limiting client participation and sparse aggregation are particularly relevant to FA, they have applications for FL problems as well.

Conclusion

We are optimistic that FL will continue to expand, both as a research field and as a set of practical tools and software systems that allow application by more people to more types of data and problem domains.

For those interested in learning more about active research directions, the recently updated *Advances and Open Problems in Federated Learning* provides a broad survey, with coverage of important topics not covered in this article, including personalization, robustness, fairness, and systems challenges.[10] If you are interested in a more hands-on introduction to FL, such as trying out algorithms in a simulation environment on either your own data or standard datasets, the TensorFlow Federated tutorials[b] are a great place to start—they can be executed and modified on the fly in the browser using Google Colab.

Acknowledgments

The authors thank Alex Ingerman and Marco Gruteser for their helpful feedback, as well as the many people at Google who have helped develop these ideas and bring them to practice.

b https://www.tensorflow.org/federated/tutorials/federated_learning_for_image_classification

References

1. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conf. Computer and Communications Security*, 308–318; https://dl.acm.org/doi/10.1145/2976749.2978318.
2. Balle, B., Kairouz, P., McMahan, H.B., Thakkar, O., Thakurta, A. Privacy amplification via random check-ins. arXiv, 2020; https://arxiv.org/pdf/2007.06605.pdf.
3. Bell, J.H., et al. Secure single-server aggregation with (poly)logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conf. Computer and Communications Security*, 1253–1269; https://dl.acm.org/doi/10.1145/3372297.3417885.
4. Bittau, A., et al. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symp. Operating Systems Principles*, 2017,441–459; https://dl.acm.org/doi/10.1145/3132747.3132769.
5. Bonawitz, K., et al. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conf. Computer and Communications Security*, 1175–1191; https://dl.acm.org/doi/10.1145/3133956.3133982.
6. Bonawitz, K., et al. Towards federated learning at scale: system design. In *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019; https://arxiv.org/pdf/1902.01046.pdf.
7. Carlini, N., Liu, C., Erlingsson, U., Kos, J., Song, D. The secret sharer: evaluating and testing unintended memorization neural networks. In *Proceedings of the 28th Usenix Security Symp. 2019*, 267–284; https://dl.acm.org/doi/10.5555/3361338.3361358.
8. Carlini, N., et al. Extracting training data from large language models. arXiv, 2020; https://arxiv.org/abs/2012.07805.
9. Dwork, C., McSherry, F., Nissim, K., Smith, A.D. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Intern. Assoc. Cryptologic Research Theory of Cryptography Conf.*, 2006, 265–284. Springer-Verlag; https://iacr.org/archive/tcc2006/38760266/38760266.pdf.
10. Kairouz, P., et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning 14*, 1–2 (2021); https://arxiv.org/abs/1912.04977.
11. Kairouz, P., Liu, Z., Steinke, T. The distributed discrete Gaussian mechanism for federated learning with secure aggregation. In *Proceedings of the 38th Intern. Conf. Machine Learning 139* (2021), 5201–5212; http://proceedings.mlr.press/v139/kairouz21a/kairouz21a.pdf.
12. Kasiviswanathan, S.P., Lee, H.K., Nissim, K., Raskhodnikova, S., Smith, A. What can we learn privately? *SIAM J. Computing 40*, 3 (2011), 793–826; https://dl.acm.org/doi/10.1137/090756090.
13. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., Agüera y Arcas, B. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th Intern. Conf. Artificial Intelligence and Statistics*, 2017, 1273–1282; http://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf.
14. McMahan, H. B., Ramage, D., Talwar, K., Zhang, L. Learning differentially private recurrent language models. In *Proceedings of the 2018 Intern. Conf. Learning Representation*; https://openreview.net/pdf?id=BJ0hF1Z0b.
15. McMahan, H.B., Ramage, D. Federated learning: Collaborative machine learning without centralized training data. Google AI Blog (Apr. 6, 2017); https://ai.googleblog.com/2017/04/federated-learning-collaborative.html
16. Paulik, M., et al. Federated evaluation and tuning for on-device personalization: system design & applications. arXiv, 2021; https://arxiv.org/abs/2102.08503.
17. Ramage, D., Mazzocchi, S. Federated analytics: collaborative data science without data collection. Google AI Blog (May 27, 2020); https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html.
18. Ramaswamy, S., et al. Training production language models without memorizing user data. arXiv, 2020; https://arxiv.org/abs/2009.10031.
19. Subramanyan, P., Sinha, R., Lebedev, I., Devadas, S., Seshia, S.A. A formal foundation for secure remote execution of enclaves. In *Proceedings of the 2017 ACM SIGSAC Conf. Computer and Communications Security*, 2435–2450; https://dl.acm.org/doi/10.1145/3133956.3134098.

Kallista Bonawitz (Cambridge, MA, USA), **Peter Kairouz** (Seattle, WA, USA), **Brendan McMahan** (Seattle, WA, USA) and **Daniel Ramage** (Seattle, WA, USA) are researchers at Google, focusing on decentralized and privacy-preserving machine learning. Their team pioneered the concept of federated learning (https://bit.ly/3xUOs6L) and continues to push the boundaries of what is possible when working with decentralized data using privacy-preserving techniques.