# Federated Computation: A survey of concepts and challenges

Akash Bharadwaj[*] Graham Cormode[†]

**Abstract**

Federated Computation is an emerging area that seeks to provide stronger privacy for user data, by performing large scale, distributed computations where the data remains in the hands of users. Only the necessary summary information is shared, and additional security and privacy tools can be employed to provide strong guarantees of secrecy. The most prominent application of federated computation is in training machine learning models (federated learning), but many additional applications are emerging, more broadly relevant to data management and querying data. This survey gives an overview of federated computation models and algorithms. It includes an introduction to security and privacy techniques and guarantees, and shows how they can be applied to solve a variety of distributed computations providing statistics and insights to distributed data. It also discusses the issues that arise when implementing systems to support federated computation, and open problems for future research.

**Keywords:** federated learning, federated analytics, histograms, mean estimation, differential privacy, secure multiparty computation

**Competing Interests:** The authors are employees of Meta Platforms, Inc ("Meta").

## 1 Introduction

Modern data management systems make extensive use of large volumes of data to provide services to their users. These functionalities range from providing simple statistics and descriptions of data, through more involved queries, to complex machine learning models that can make recommendations and suggestions. The data that informs these applications is typically drawn from a range of sources, but many of these rely directly on the information of individual data subjects: people's preferences and actions. This information is potentially very sensitive, as it can relate to individuals' medical information (e.g., their diagnoses, medication, family history); finances (e.g., their salary, savings, debts, payment history); and other highly personal information (e.g., their interests, political and religious beliefs, sexuality).

As a result, there are increasing numbers of legal protections and regulations being instituted to control the collection and processing of personal data. Some prominent examples include: GDPR, a data protection regulation that has been enforced in Europe since 2018 and is the template for regulations in many other jurisdictions; ePD, an ePrivacy regulation on electronic communications in Europe that is currently being introduced; and CCPA, the Caifornia Consumer Privacy Act that has been in effect in the US state of California since 2020. Simultaneously, many companies are also making promises of privacy in their products via technical means, such as end-to-end encryption in messaging applications (WhatsApp, Signal), and differential privacy implemented in Apple's iOS since version 14. These competing drives give the appearance of an impasse: modern data applications benefit significantly from being informed by personal data, but regulations and commitments seem to preclude the direct usage of private data for these purposes.

The resolution of this conundrum is the careful adoption of a range of mitigations, both technical and non-technical in nature. The key technical mitigations come in the guise of various forms of Privacy Enhancing Technologies (PETs), which limit the information that can be exposed as a result of the computation. In conjunction with transparent privacy policies, and with the consent of users, these technologies enable users to benefit from the positive outcomes of modern models of data, while ensuring that privacy risks are strictly controlled.

[*]Meta, Menlo Park, CA.

[†]Meta, Coventry, UK. gcormode@meta.com,

**Outline.** In the remainder of this section, we describe the model of federated computation in more detail, and the main considerations for this kind of privacy-preserving data processing. In Section 2, we spell out the mathematical definitions needed to understand the privacy and security tools adopted within federated computation. These are used in Section 3 to build practical algorithms for gathering statistics in the federated model. We go on to discuss the practical considerations in building and deploying federated computations in Section 4. Section 5 presents further issues and open problems from the community to tackle; we conclude with Section 6. The outline and content of this presentation follows closely a tutorial presented by the authors at the SIGMOD and Web conferences in 2022 [Bharadwaj and Cormode, 2022], which may be viewed in conjunction with this survey article.

**Federated Computation Overview.** In this survey, we focus on a recently emerged class of PETs, referred to as Federated Computation. The essence of Federated Computation is that data remains in the control of the data subject (typically, on a personal mobile device belonging to the user, or within a secure database), and the flow of information about the user to a computation is restricted to provide strong privacy and security guarantees. As a model of computation, federated computation can be compared to the popular MapReduce model of parallel, distributed computation. These two models have several features in common. The data is massively distributed — in the federated setting, we are potentially dealing with up to billions of user devices. The instructions on what computation to perform is sent out to the devices, rather than having the data gathered up centrally. But, in contrast to MapReduce and other models of distributed computation, there are several features of federated computation that make it distinct and challenging. Consent, privacy and security are first order concerns in the federated setting. Hence, data is retained under the control of users, and extra attention is paid to privacy and security implications of the communications between the users and the central computing entity, which we refer to as the server. There is considerable heterogeneity across many aspects of the computing environment: the user devices may vary considerably in their compute power and their communication bandwidth. They may have limited memory and storage capabilities, and so we should design protocols to be very lightweight. The connectivity of devices may be intermittent and unreliable. Lastly, the nature of data itself held by each user may vary considerably across the user population, in terms of the distribution of values held, and the volume and stability over time of the data values. Collectively, these pose novel challenges for algorithm designers and system builders who want to work in the federated setting.

**Privacy Principles.** Different PETs exhibit different privacy principles and emphasise a variety of objectives. The principles most relevant to federated computation can be articulated as follows:

- **Data minimization** aims to gather the least amount of data necessary to perform the desired computation.
- **Purpose limitation** ensures that the data gathered is only used for a specific, limited purpose.
- **Ephemerality** requires that the data is not retained once its purpose have been met.
- **Access controls and security** should be put in place to ensure that data is only accessible to legitimate entities for approved purposes, along with provenance tracking to help debug issues that may arise.
- **Aggregation** is used so that only broad population level statistics are generated and stored, rather than data pertaining to individuals.
- **Anonymization** requires that data is handled so that information cannot be traced back to individual users.
- **Plausible deniability** mandates that information revealed by users introduces a level of uncertainty about the accuracy of each disclosure.

**Private Computation models.** There are a wide range of dimensions across which to study computation over private data, depending on the nature and sensitivity of the data being handled. Different instances of federated computation instantiate various of these dimensions.

- **Level of centralization.** Is the data gathered centrally, or processed by distributed entities? In between, there can be hierarchical models, where data is partially aggregated by intermediaries before being passed to a server to complete the analysis. Federated computation tends to involve a high level of distributed processing.

- **Nature of privacy guarantees.** Is the data protected by enforcing (syntactic) rules on truncation of data, or (statistical) processes that add random noise? Federated computation efforts mostly adopt the current focus on statistical noise.
- **Type of data handled.** Does the computation apply to data that is tabular, image, video or graph-based?
- **Type of task handled.** What is the computation of interest — to materialize some statistics, train a machine learning model, generate synthetic data?
- **Granularity of privacy.** Is the definition of privacy applied to the individual record level, to an individual device, to a user's entire presence across a collection of devices, or to a larger group? All are valid choices, but we require a greater level of distortion to protect larger footprints in the data.
- **Nature of data processing.** Is the data treated as a finite, static data set, or as an infinite growing stream of records?
- **Nature of query reporting.** Are the outputs of the computation released as a one-shot result, or as longitudinal observations tracked over time?
- **Scale.** How many users contribute their data, and how much data do they each possess? It is often easier to achieve a better privacy-accuracy tradeoff at larger scales, since the amount of noise needed to hide the presence of an individual can be proportionately smaller. Scale can also affect which technologies are appropriate, since e.g., secure hardware might hit capacity limits, while some security protocols have computational cost that scales quadratically with the number of participants.
- **User availability.** Can data disclosures be obtained on-demand uniformly at random, or under a more restricted access pattern?

The federated approach then instantiates some of these dimensions. Data remains under the control of users by being held on their devices devices. Only a small amount of necessary data is shared with central servers. All communication is done with strong security guarantees to prevent eavesdropping (i.e., we should ensure that all channels are encrypted). Then additional privacy guarantees may be provided on top of this skeleton based on the adoption of other privacy-enhancing technologies, such as random noise addition, anonymous communication channels, and secure aggregation primitives, described in more detail in Section 2. In summary, we seek instantiations of federated computation that are private, secure and distributed. This helps delineate the federated model from prior computational models, which have sought subsets of these properties, but rarely achieved all three at once.

**Thumbnail sketch of a federated computation.** Figure 1 shows a schematic of how a federated computation might proceed, taking advantage of multiple privacy technologies along the way, such as secure multiparty computation (SMC) and differential privacy (DP), described below. In what follows, we focus on the actions of the client devices (or clients, for short) that perform the protocols on behalf of one or more users. Each client device receives an invitation from a coordinating server (the server, for short) to participate in a federated computation task. This is shown in Figure 1 with a dashed line from server to clients. The client may opt-in or opt-out depending on the device privacy policy and the device user's expressed preferences. For a client that participates, the description of the task from the server may include parameters that inform how to apply an algorithm on device to data stored there. This results in some partial outputs, computed entirely on the device. This step embodies the first privacy principle, to perform on-device data management and manipulation. Before the outputs are released for upstream processing, the client may perturb them, for example by introducing statistical noise to meet some level of differential privacy. Messages from clients to the server are then gathered via a secure infrastructure, illustrated as a solid line from clients to aggregator and server in Figure 1. This may use security technologies to ensure, for instance, that the server is not able to associate any message with any user identifier — that is, the infrastructure implements anonymous channels between clients and server. Aggregation may be used to ensure that that server only sees the aggregation of the client messages, rather than the individual unaggregated responses. The server then performs final aggregation and processing of these received messages, in order to produce the final output of this round of federated computation. This output may be delivered to a downstream processor, or could inform additional rounds of federated computation, such as optimization of a loss function, or refinement of an approximate statistic. Thus, the combination of these technologies give an overall federated computation instance that includes multiple pieces to restrict the flow of data between the clients and the server.

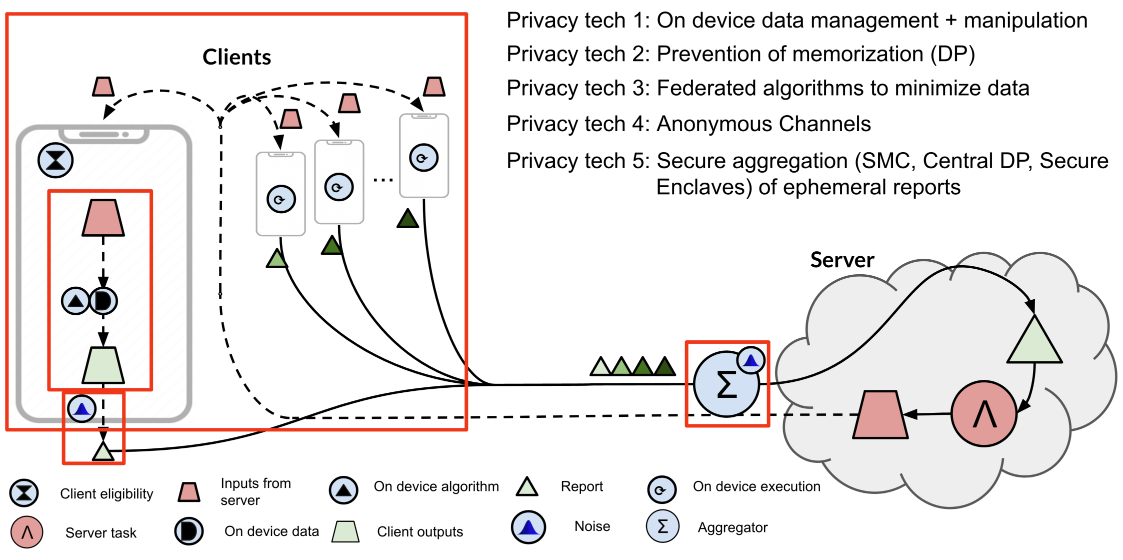
Clients
Privacy tech 1: On device data management + manipulation
Privacy tech 2: Prevention of memorization (DP)
Privacy tech 3: Federated algorithms to minimize data
Privacy tech 4: Anonymous Channels
Privacy tech 5: Secure aggregation (SMC, Central DP, Secure Enclaves) of ephemeral reports
Server
Client eligibility
Inputs from server
On device algorithm
Report
On device execution
Server task
On device data
Client outputs
Noise
Aggregator

Figure 1: Schematic of a private computation using federated computation principles

**Horizontal and Vertical Federation.** An important distinction drawn in the literature is whether the data is federated along horizontal or vertical lines. We imagine the full collection of data to be represented by a massive table, where every row of data corresponds to an individual, and every column corresponds to a feature. If we divide this table with vertical lines, and allocate these slices to clients, then we have the case where the same set of individuals is represented across multiple clients, so that each client holds a different set of information about the set of users. This captures scenarios where data is split across a number of silos. This could be a medical setting, where each client is a hospital department (e.g., cardiology, oncology) holding information relevant to that speciality about a common population of patients, working together to better understand a new disease. Or it could be an online advertising setting where each client is a separate advertiser with data on how different individuals have interacted with their ads.

If, on the other hand, we divide the data table horizontally, then each client holds data on non-overlapping collections of individuals against the same schema. This could describe the case where each client is a mobile device with information on one individual's browsing history, where the goal would be to improve performance and pre-fetching. Or it could capture the setting where each client is a bank that holds the financial data of multiple individual accounts, and the banks want to cooperate to learn patterns of fraudulent activity.

Scenarios with a moderate number of participants (e.g., banks, hospitals), each representing many individuals, are often referred to as "cross-silo", whereas scenarios with a very large number of participants representing one or a few individuals are referred to as "cross-device", since these usually map onto cases where a large number of mobile devices are participating in the computation. Note that across all these scenarios there is a considerable range of computational power and other resources: we expect a smartphone to be much weaker than a bank's computer! In what follows, our focus is on the cross-device setting, as the largest challenges of scale and heterogeneity arise here. Cross-device settings typically entail horizontal federation: each device holds data on the device's user(s) drawn on the same set of features.

Here, the challenges for designing and deploying federated computation are twofold. First, there is the scale of the data. The volume of data can be truly mind-boggling, for instance in cases where video data stored on the clients will be used to train machine learning models. In these settings, it simply may not be practical to centralize the volume of data involved, let alone the questions of legality and permissibility of centralization. It is therefore necessary to design workflows that will process the data 'at rest' on each client device, in order to extract information about a smaller volume of higher-order features to contribute to the subsequent processing. The second challenge is the need to offer strong privacy guarantees. Keeping the data on device is an important step in this regard: the clear promise that their data remains on their device at all times can provide strong and intuitive reassurance to users. However, while it is certainly the case that the communication from client to server is less revealing than the totality of the user data used to form the response, there is still the potential for leakage, and additional steps are needed to protect the content of the messages. This is in addition to protections that are needed to secure the data on device.

**Types of Federated Computation.** In the literature to date, there have been two main categories of tasks that have attracted attention. 'Federated learning' (FL) refers to tasks tightly focused around training machine learning models over distributed data. This focus emerges due to the high importance of machine learning as an enabler of cutting-edge applications across the high tech industry, achieving state-of-the-art performance on many text and multimedia based tasks. As a result, there is a surge in interest among the research community, with many thousands of scientific papers on federated learning published each year. The essence of federated learning is that the server will share a candidate model to a batch of clients, each of whom will apply the model to their data examples, and propose improvements to the model based on how it performs on their data. This procedure is then repeated over multiple rounds to converge on a model that can be released more widely. Different approaches to FL vary based on how they determine the model architecture and its hyperparameters; how they handle heterogenous data and unreliable clients; how the model parameters are optimized, and how the client batches are chosen; ways to compress the models or the model updates; and many more aspects.

Other applications of federated computation are still emerging. The term 'federated analytics' (FA) has been proposed [Ramage and Mazzocchi, 2020] as a counterpoint to federated learning, to address the variety of cases where the objective is to gather statistics on distributed data in order to describe properties of the data distribution. Although there are many points of similarity between federated learning and federated analytics, there are several qualitative differences. FL tends to operate over very many rounds, while FA tasks are often completed in one or a few rounds. FL batches are typically comprised of a few hundreds to thousands of clients, while FA tasks can call on tens of thousands to millions of clients to participate. FL may require clients to perform backpropagation on models that are tens of megabytes in size, while FA may only demand simple computations on data summaries that are kilobytes in size. Hence, there are meaningful differences in the types of algorithms and operation between these two notions of federated computation. In this survey, we will present examples of recent efforts in federated computation that address both federated analytics and federated learning scenarios. We provide more coverage of FA than FL, and refer the interested reader to a comprehensive survey on federated learning for a corresponding study of that topic [Kairouz et al., 2021]. At this time, the notions of FA and FL are sufficient to capture the vast majoritiy of work under the banner of federated computation, but we can anticipate that in future other examples will emerge that fall outside of these two categories.

**Drawbacks and Limitations.** While we position federated computation as a compelling model for allowing the processing of private data at scale, it is not without its potential drawbacks and limitations. First, we should ask whether federated approaches are fair in the ways that they model user populations. There is an inherent bias in who is represented within federated computations: only people with sufficiently powerful mobile devices may be included in a federated computation, say. Users of older devices, or without their own device, may be excluded from the computation, and hence not represented in the output. This may skew the information produced, which may be used more widely in recommendation or decision making tasks, to preferentially represent more affluent and tech savvy individuals. Second, we can ask whether the additional privacy promises of federated computation come at higher financial and societal costs. It seems likely that performing the computations on many weak devices may consume more energy overall than performing the same computation on high-end devices in a data center. This is not certain—the energy cost to communicate client information to the data center might tip the balance in the opposite direction. However, for now we lack careful accounting of the energy costs to make a meaningful comparison. Both these issues demand further attention and evaluation from the research community to quantify the impacts and propose mitigations.

**Summary of key points.** We have introduced the model of federated computation as a framework for privacy-preserving computations over collections of users at very large scale. It can be understood as a "privacy-preserving MapReduce", where clients perform the mapping functions, and servers are reducers. The intuitive privacy promise is that user data stays on client devices, and only sufficient statistics are shared with the server in adherence to privacy principles. Additional privacy comes from adoption of a collection of privacy-enhancing technologies that we will discuss in subsequent sections of this survey. To date, federated computation has been widely adopted for machine learning applications, under the banner of federated learning. The complementary suite of federated analytics to gather statistics and aggregates over client populations is emerging in parallel.

| **Technique** | **What does it do?** | **Pros** | **Cons** |
|---|---|---|---|
| Data Minimization | Share the least possible information | Intuitive to users | Lacks formal guarantees |
| Secure Multiparty Computation | Only reveal the result of computation | Strong security | Vulnerable to dropouts |
| Secure Aggregation | Only reveal the result of aggregation | More robust vs. SMC | Limited in capability |
| Syntactic privacy | Enforce conditions on the output | Intuitive to users | Vulnerable to attacks |
| DP with trusted entity | Entity adds noise to the output | Formal guarantees | Requires trust in aggregator |
| Local Differential Privacy | Clients add noise to their inputs | Stronger protection | Much more noise overall |
| Distributed differential privacy | Clients add small noise to messages | Matches central DP | Vulnerable to poisoning |

Table 1: Overview of different privacy enhancing technologies for federated computation

## 2 Technical Preliminaries

In this section, we introduce some of the technical preliminaries around the technologies that support privacy and security. We distinguish these two concepts: *privacy* refers to efforts to protect the output of a computation, while *security* is used to protect the intermediate values within a computation. In other words, when considering a computation over sensitive data, we want privacy on the result to ensure that the information revealed does not compromise the privacy of an individual, and we want security to ensure that we only learn the intended output and do not expose any partial results. In what follows, we highlight five concepts relevant to security and privacy: data minimization (Section 2.1), secure multiparty computation (Section 2.2) and its specialization to secure aggregation (Section 2.3), and both syntactic and statistical models of privacy (Sections 2.4 and 2.5). Table 1 gives a very high-level summary of each technique we discuss below; refer to the relevant section for the nuanced explanation of each snippet, which is flagged in italics within the text.

### 2.1 Data Minimization

A fundamental principle of privacy is to minimize the data that is used or shared within any computation: *share the least possible information*. Concretely, this means when calculating the average salary of a given group of individuals, there is no need to share any further information about their birth dates, home addresses, demographic characteristics etc. This is instantiated more generally in other settings by aggregations (e.g., reporting only the total number of visits a user made to a website, rather than the full details of every visit), and transformations (e.g., sending only suggested updates to an ML model, rather than all the data points that led to this conclusion). This principle is easy to grasp for may lay users, and seems intuitive (*intuitive to users*). However, data minimization alone is not sufficient to provide the strongest level of privacy protection (*lacks formal guarantees*). The minimal data can still be revealing: in the salary example, it can still be undesirable to reveal the individual salary values. More insidiously, transformed and compressed versions of the data can still be used with assumptions about the prior distribution to infer the original data values. For instance, it has been shown that the gradients computed in training a deep learning model can nevertheless allow quite accurate reconstruction of the original training example [Zhu et al., 2019]. Therefore, in what follows we will seek additional levels of protection in the federated setting. We will assume that all communications are direct between each client and the server, and that these are encrypted securely. Hence, our focus will be on what information the server can learn about the client's data, and what can be learned from the server's output about the clients.

### 2.2 Secure Multiparty Computation

In the model of Secure Multiparty Computation (SMC), we have multiple servers working together to compute some desired function without being exposed to the input values or other intermediate values. We will discuss the case when servers are honest and do not collude—-the so-called "honest-but-curious" setting (variants of this model can tolerate more adversarial actors). Data values are "shared" between servers, so that no server sees the whole message, and so only the final result of the computation is revealed: hence, it *only reveals the result of computation*.

A key concept is the idea of additive secret shares, due to Shamir [1979]. Here, an individual value $x$ — say, someone's age — is split into $m$ pieces $x^{(1)}, x^{(2)}, \dots x^{(m)}$ ("shares"), so that $\sum_{i=1}^{m} x^{(i)} = x$. This allows operations to be performed on the shares, without revealing anything about the original value $x$. We may use the shorthand $[\![x]\!]$ to refer to the set of shares, i.e., $[\![x]\!] = (x^{(1)}, x^{(2)}, \dots x^{(m)})$..

In the federated setting with a very large number of clients, it may not be feasible to have all clients participating in a secure multiparty computation: the amount of communication can be very large, and we cannot rely on all clients remaining online for the duration of the computation. Instead, a practical approach is to have a small number of non-colluding servers (say, two) who each receive input shares from all clients, then perform the computation between themselves.

A simple example of this is to compute the sum of all client input values. Suppose we have $n$ clients with values indexed as $y_1, y_2, \ldots y_n$. Each client can generate a share into two pieces by picking a random value $r_i$, and generating $x_i^{(1)} = r_i$ and $x_i^{(2)} = x_i - r_i$. Then the clients can send their shares to two collaborating servers, $x_i^{(1)}$ to server 1, and $x_i^{(2)}$ to server 2. Server 1 then forms the sum $S_1 = \sum_{i=1}^{n} x_i^{(1)}$, and Server 2 forms the sum $S_2 = \sum_{i=1}^{n} x_i^{(2)}$. Note that $S_1$ and $S_2$ individually reveal nothing about the inputs: SMC provides *strong security*. Finally, the servers can combine their sums to compute

$$S = S_1 + S_2 = \sum_{i=1}^{n} r_i + \sum_{i=1}^{n} x_i - r_i = \sum_{i=1}^{n} x_i,$$

as required.

This example gives the intuition but skimps on some details. For this to work formally, we require that the input values $x_i$ and random values $r_i$ be represented within a finite field, e.g., the integers modulo a prime number. This technicality ensures that the shares truly reveal no information about the input that the encode.

It is natural to ask what set of operations can be computed in the SMC model. The answer depends on how much extra work and how many rounds of communication we want to tolerate. Some functions come immediately. We can write $[\![x]\!] + [\![y]\!] = [\![x + y]\!]$: we can sum two shares (entry-wise) to obtain the share of their sum. Similarly, subtraction follows as $[\![x]\!] - [\![y]\!] = [\![x - y]\!]$, and multiplication by a (public) scalar value $c$ satisfies $c[\![x]\!] = [\![cx]\!]$.

Other operations are more involved. Multiplication of two private values $[\![x]\!]$, $[\![y]\!]$ can be done with some interaction if we have access to some "helper" values $(a, b, c)$ with the property $ab = c$ — this is the so-called 'Beaver triples' method of multiplication [Beaver, 1991]. Non-linear functions, such as comparisons (greater than, less than) can be done in various ways, such as by converting the shares from 'arithmetic shares' to 'binary shares' (shares of the binary representation of integers), and operating on the binary shares [Damgård et al., 2006, Nishide and Ohta, 2007]. A full coverage of working within the secure multiparty computation model is beyond the scope of this survey (instead, see [Evans et al., 2018]). For many basic tasks, simple arithmetic shares suffice. There is still a tradeoff here: more specific SMC protocols for particular tasks can be more efficient but less flexible than more general-purpose approaches.

A more tangible limitation is the fact that SMC algorithms may be badly affected if clients drop out part way through the protocol, even if the clients are only needed to initiate the protocol: it is *vulnerable to dropouts*. For instance, suppose that in the summation protocol above, client $i$ sends $r_i$ to server 1, but then crashes, and does not send the other share to server 2. Leaving $r_i$ in the final summation would lead to a meaningless output, since we should expect the values of $r_i$ to look like arbitrary large integers. Other simple attempts to patch this, say by having a intermediate server receive both shares from clients, or having the servers compare lists of which clients have sent shares, inevitably weaken the security properties of the protocol, or require stronger trust from the participants. Instead, in the next section we consider ways to use SMC-like techniques but with only a single aggregating server.

## 2.3 Secure Aggregation

Secure Aggregation refers to a primitive which computes the sum of values from a large cohort of clients. That is, it performs the basic summation task described in the previous section, and *only reveals the result of the aggregation*. Different instantiations of Secure Aggregation operate in different security models with different trust assumptions. The idea is that the task of summation is sufficiently ubiquitous that it is worthwhile building an optimized protocol to handle this foundational task. Consequently, the secure aggregation protocol can be made *more robust than general purpose SMC*, at the cost of being *more limited in capability*. Some different approaches include:

- The SMC-based approach from the previous section making use of $k \geq 2$ servers.
- A trusted computation approach, where a trusted enclave sends a 'mask' (random value) to each client. Clients apply the mask to their inputs by computing the sum of their mask and their value, and send this to the server. The server also receives the sum of all the masks of participating clients from the trusted enclave, and subtracts this from the sum of their messages to obtain the correct sum.

- A subset of clients cooperate amongst themselves to perform crpytographically secure aggregation without any server involvement [Roth et al., 2019]
- Clients perform secret sharing of their value with all other clients, and all pass the shares they have received to a server who sums them to get the true sum [Bonawitz et al., 2017].
- Clients secret share to a small randomly chosen subset of other clients. All shares are combined by one server to get the sum [Bell et al., 2020].

To make these protocols practical, it is necessary to understand how robust they are to clients dropping out during the execution of the protocol. They also vary in how much trust is assumed between the different participants, and what level of connectivity is required: is it assumed that any pair of clients can communicate directly, and make use of a public key infrastructure to send their messages securely? In Section 3.1.1 we present one such Secure Aggregation protocol in more detail, and describe its strengths and weaknesses.

## 2.4 Syntactic Privacy

Early efforts to provide privacy guarantees on data outputs attempted to define and *enforce syntactic conditions that should hold on the output data* in order to make it private. For instance, the notion of $k$-anonymity says that each record in the output should be supported by at least $k$ individuals [Samarati and Sweeney, 1998]. Consider an application that published a heat map of user's home addresses, by dividing a map into non-overlapping regions and recording the number of users in each cell. Some cells in this map might be almost empty, corresponding to a very small number of individuals, and posing a potential privacy threat by leaking locations of isolated individuals. Applying the requirement of $k$-anonymity to this data would entail ensuring that each cell has at least $k$ individuals associated with it. This could be achieved by merging together sparse cells until the merged cells pass the $k$ individual threshold, or else by simply by 'blanking out' any cells with fewer than $k$ individuals.

This notion of $k$-anonymity has an *intuitive appeal to users*, and is often used as a first step towards privacy. However, on its own, $k$-anonymity is not robust against sophisticated attackers, and has been shown to be *vulnerable to a number of attacks*. Sometimes, knowledge of the procedure used to obtain $k$-anonymity can be used to partially reverse the aggregation [Machanavajjhala et al., 2007]. In other cases, $k$-anonymity does not protect an individual if the other individuals they are grouped together with all share some sensitive features. For these reasons, $k$-anonymity (along with other syntactic approaches to privacy) has been largely deprecated as a meaningful privacy protection. Nevertheless, it can still be a useful stepping stone: data that is released with both $k$-anonymity and some stronger privacy guarantee provides both the intuitive "safety in numbers" assurance of $k$-anonymity, along with some more precise protections. These formal privacy definitions are mostly in the form of statistical guarantees, as described in the next section.

## 2.5 Statistical Privacy via Differential Privacy

A natural approach to providing privacy on numerical data is to add random noise to the result—enough to introduce some uncertainty, but not so much as to distort the message of the data. The noise can be explicit (noise drawn from an appropriate random distribution) or implicit (noise introduced via randomly sampling clients to participate in the computation). Noise addition is at the heart of statistical models of privacy, the most prominent of which is Differential Privacy (DP). Here, we give a brief overview of the main concepts — for a full study, see the textbook of Dwork and Roth [Dwork and Roth, 2014]. An informal statement of differential privacy is that for a (randomized) algorithm that operates on sensitive data, we should have that the otuput distribution looks similar when an individual user's contributions are added or removed. To formalize this, we consider a (randomized) algorithm $A$ operating on two similar datasets $D$ and $D'$. $D$ and $D'$ are said to be "neighboring" if they differ only in the information of one individual. Then, for any set of possible outputs, we require that

$$\Pr[A(D) \in O] \leq \exp(\epsilon) \Pr[A(D') \in O] + \delta$$

where $\epsilon$ and $\delta$ are privacy parameters. We should think of $\epsilon$ as not too large, and $\delta$ as being very small. Then $\exp(\epsilon)$ is not much bigger than 1, and this definition can be interpreted as saying that the probability of seeing an output in the set $O$ should look about the same whether the input was $D$ or $D'$. Here, $\epsilon$ and $\delta$ control the 'wiggle room' for how tight the similarity requirement is. Concretely, we consider $\epsilon \leq 0.1$ to correspond to very high privacy, while $\epsilon > 1$ is a lower privacy promise.

This definition gives a property that may or may not hold for any given algorithm. The next question is whether this definition can be satisfied by some algorithm for a specific task. Fortunately, there are some generic recipes for obtaining differential privacy (as well as more sophisticated algorithms tuned to particular tasks). In particular, given a function of interest $f(D)$ that can be applied to a data set $D$ to obtain some numerical output, we can define the 'sensitivity' of $f$ as the maximum change in value that can occur as the data set is varied. That is, define $\Delta(f) = \sup_{D,D'} \|f(D) - f(D')\|$, where $D$ and $D'$ differ in only one individual, i.e., they are neighboring inputs as discussed above. The norm $\|\cdot\|$ is typically the $L_1$ or $L_2$ (Euclidean) norm.

Given this set up, we can obtain a differentially private algorithm by computing $f(D)$ exactly on our input $D$, then adding random noise scaled by $\Delta(f)$ and the target privacy level $\epsilon$. The random noise can be from an appropriate statistical distribution, such as the Laplace distribution or the Gaussian, depending on exactly what nature of DP guarantee is desired. To begin with, we assume noise drawn from a continuous distribution, but subsequently we describe discrete noise distributions that are needed to meet constraints in the federated setting.

The procedure described so far is essentially a centralized one. That is, we assume that we have collected all the sensitive data $D$ together in one location, and so can compute $f(D)$ easily before adding the noise. When we turn to the federated setting, the picture is less clear. There are two intertwined challenges to overcome: (1) how to compute the value of the function $f$ over inputs that are distributed among many clients and (2) how to ensure that sufficient noise is added without revealing the true (noiseless) value of the function $f$ to any party?

There are several ways to tackle these questions, and we highlight three main options in the following subsections.

### 2.5.1 Differential Privacy with a Trusted Entity

The simplest way to achieve differential privacy in the federated setting is to commission a trusted entity to receive the input data from the clients, and perform the aggregation and noise addition before passing the results on to the server for downstream processing. That is, we address the challenge of distributed data by centralizing the privatization process, and engage a *trusted entity to add noise to the output*. The benefits of this approach are the relative simplicity of the algorithmic work, and hence the (relative) ease of implementation in order to obtain the *formal guarantees of differential privacy*. The drawbacks are obvious: *we have to place trust in this entity* not to 'snoop' on the data that they are processing. However, there are advantages of this approach compared to the naive approach of simply sharing data from clients directly to the server. By giving the trusted entity one task to do (produce differentially private output), we reduce the requirements, and make this component more amenable to scrutiny and auditing. It is important to draw the circle of trust as tightly as possible, to avoid the potential for unwanted privacy leaks. Servers that know even some seemingly innocuous information (say, knowing which clients participated in a computation, but not anything about the contents of their data) can nevertheless derive some knowledge from this exposure. To this end, we can implement the trusted entity on special hardware, such as a trusted execution environment (TEE), aka, a secure enclave [Peisert, 2021]. This is a server-like machine that protects the data that it is processing by working with encrypted memory and storage. It executes code via a special processor mode that can provide cryptographic attestation to the client that only the intended computation was performed, and no information about their data was shared with any other party. Further protections can be introduced, such as using additional security primitives to provide "anonymous channels" from clients to the trusted entity, so that there is no way for the entity to link the messages it receives back to any client.

### 2.5.2 Local Differential Privacy

Another approach for each client to ensure that differential privacy holds is for *the clients themselves to introduce the DP noise*. That is, when formulating their messages to the server providing their input to the aggregation, each client perturbs it via a randomized algorithm so that the DP definition holds: given an output (noisy) message, there is sufficient uncertainty as to what input this resulted from. Hence, the client does not need to trust any other entity in order to be assured that the privacy guarantee has been met, leading to a *strong model of protection*. This paradigm is known as local differential privacy (LDP), since the privacy is applied locally to the client, rather than centrally. There are a wide range of ways to achieve LDP, based on different representations of data and different schemes for different tasks, so we refer the interested reader to other surveys on this topic [Yang et al., 2020, Wang et al., 2020, Cormode et al., 2018].

The central technique within local differential privacy is the notion of Randomized Response (RR) [Warner, 1965]. This is a simple way to perturb a single binary value. Under randomized response, given an input bit $b$, with a fixed probability $p > \frac{1}{2}$ the client will output $b$, otherwise it outputs $1-b$. This provides an $(\epsilon, 0)$-DP guarantee on this output procedure, where the choice of parameter $p$ determines the value of $\epsilon$: $p$ close to 1 gives weaker privacy (higher $\epsilon$), while $p$ close to $\frac{1}{2}$ gives stronger privacy (lower $\epsilon$). The differential privacy constraint requires that $(1-p) \leq \exp(\epsilon)p$, so rearranging this, the precise relationship is given by $p = \exp(\epsilon)/(\exp(\epsilon)+1)$. A server who receives many output bits noised by randomized response can use this information along with knowledge of the parameter $p$ that was used in order to estimate the fraction of input bits that were 1. More complex algorithms are constructed by applying randomized response, or generalizations of this idea, to different aspects of the client's input. We will give a more advanced example of LDP noise addition in Section 3.1.2.

The chief drawback of LDP is that it introduces *a lot of noise* into the computation. Essentially, each client is independently adding the same amount of noise as would be added (once only) in the centralized case. When the noise is chosen so that it is represented by an unbiased random variable, there is some cancellation. Still, when adding noise for $n$ clients, the total variance of noise added is $n$ times larger than in the centralized case, and hence the typical error (corresponding to the standard deviation of the noise) is $\sqrt{n}$ times larger, which can be substantial since we expect $n$ to be large.

Local differential privacy is of particular interest within federated computation since the most prominent public examples of federated analytics have made use of LDP to gather data. Google's Rappor system was embedded within the Chrome browser to privately gather statistics on browsing behavior via randomized response [Erlingsson et al., 2014]. Operating systems from Apple and Microsoft have both used LDP techniques (building on randomized response) to collect information on how apps are used: what words are typed on smart keyboards, and how long apps are used for [Ding et al., 2017, Differential Privacy Team at Apple, 2017]. Meanwhile, Snap has used LDP to instantiate machine learning models for spam detection [Pihur et al., 2018]. These are prominent examples of federated computation in the real world. However, the trade-off of reduced trust comes at the cost of reduced utility, so that only strong signals can be detected with very large user populations participating in the data collection. As a result, the next generation of federated analytics systems may look to other ways to achieve privacy guarantees.

### 2.5.3 Distributed noise generation

Distributed noise generation aims to provide a compromise between trusting an entity to add all the noise, and LDP where every client adds a large amount of noise. Instead, *each client adds a small amount of noise to their message*, so that the aggregation of their noisy messages provides the desired result with the right amount of noise [Dwork et al., 2006]. Putting this into practice can be delicate to engineer. As a simple example, suppose that we wish to create the sum of values held by $n$ clients, where each client holds either a 1 or a 0 value — we can think of this as holding a vote to see how many clients are in favor of some option. If we gathered the data centrally, we would add noise from a Gaussian distribution with mean zero and variance $\sigma^2$ to the sum of inputs, for an appropriate choice of $\sigma$. Instead, we could ask each client to add noise from a Gaussian distribution with mean zero and variance $\sigma^2/N$ to their input. Taking the sum of all client inputs results in the true sum, plus the sum of $N$ Gaussian distributions with variance $\sigma^2/N$. This sum of $N$ Gaussian distributions is equivalent in distribution to one distribution with mean zero and variance $\sigma^2$. That is, we obtain a result that is *identical (in distribution) to performing the noise addition centrally*.

Note though that since the individual messages from each client only have a small amount of noise added, they do not offer a strong DP guarantee in isolation. The benefit of distributed noise generation is most pronounced when the messages are not visible, and only the sum of all the messages can be observed. This is an instance of a secure aggregation problem (Section 2.3): we want to reveal the sum of a collection of inputs. We can therefore achieve this using a suitable secure aggregation protocol over the client reports.

Putting this into practice requires some details to be attended to. Many secure aggregation protocol operate over discrete input values, rather than continuous, so we may want to find a discrete noise distribution that has the same additive properties. For the correct level of privacy to be achieved, we have to have sufficient number of clients participating in the protocol. That is, if the noise level is calibrated for $n$ clients to take part, but only a small fraction of these end up participating, then the resulting level of noise will be too low, corresponding to a reduced level of privacy. This is a concern if many clients may drop out of the federated computation. Conversely, if many more than $n$ clients contribute to the computation, then the total noise added will be much larger than needed. As a result, the server managing the computation will need to set up the computation with a good foreknowledge of the number of clients $n$ that will participate, and be prepared to abort (and not reveal the result) if much fewer than $n$ join. Last, we

| **Histogram technique** | **What it does** | **Pros** | **Cons** |
|---|---|---|---|
| Secure aggregation | Sum up sparse vectors | Security guarantees | No inbuilt privacy |
| LDP oracles | Perturb user inputs | Privacy guarantees | Poor utility |
| Distributed DP | Sum vectors with noise | Central DP guarantee | Need enough clients |
| Sampling-based | Sample a set of clients | Simple to implement | Trust assumptions |

Table 2: Summary of approaches to federated histogram computation

need to ensure that the noise added by clients is sampled correctly, and is not adversarially chosen to skew the result of the computation. Various techniques such as zero-knowledge proofs may be used to reduce this *vulnerability to poisoning* [Corrigan-Gibbs and Boneh, 2017].

## 2.6 Technical preliminaries summary

Building a federated computation system relies on the careful combination of security and privacy primitives. As summarized in Table 1, each technique has some limitations, and no one method may provide a full range of protections. Instead, we should design systems that incorporate multiple privacy enhancing technologies to provide more holistic protection of private information. Security primitives such as secure multiparty computation and secure aggregation protect the inputs to the computation, while privacy primitives such as differential privacy and $k$-anonymity aim to protect the outputs of the computation. Policy decisions are needed to decide what trust assumptions are made, particularly between central differential privacy applied by a trusted entity, local differential privacy applied by each client, or distributed differential privacy applied by all clients. A typical federated computation implementation will make use of multiple privacy and security technologies, such as the combination of distributed differential privacy with secure aggregation.

# 3 Federated Algorithms

To solve practical federated computation questions, we will need to make use of the various technical primitives we have discussed in order to solve more complex tasks. Each task may need a bespoke algorithm, since we currently lack general-purpose systems that can take a high-level description of a task and map it into a deployable code. In this section, we will describe a few canonical tasks—histogram construction and estimation of the means of scalars and vectors—and give examples of different federated approaches to solve them, with different tradeoffs of privacy, trust, accuracy and computational cost (time and communication). These tasks are important primitives in fededated computation, and have been used to tackle a number of problems within both federated analytics and federated learning.

## 3.1 Private Histograms

Private histograms are the bedrock of many private data analysis tasks. They solve an abstract problem that can be used as the basis to answer a variety of different queries. The basic form of the histogram problem is where each client holds a value drawn from a finite domain of $d$ possible values, $\mathcal{D}$. The goal is simply to provide a list of the aggregated counts for each label. As a concrete example, consider a setting where each client records information on a user's favorite color. Some users might prefer red, others blue or yellow. The output should record how many choose each option — i.e., how many users chose blue as their favorite color? In some cases, the domain size $d$ may be very large, and it is OK to omit options that are not selected. That is, if no one chooses 'puce' as their favorite color, we do not need to explictly output a count of zero for this option.

Histograms are most directly applicable for federated analytics tasks, tabulating the number of occurrences of different events, and supporting distributional queries. They are also foundational for federated learning, since histograms support capturing the statistical distribution of features, which is a key step in many machine learning tasks. We discuss these applications in more detail in Section 3.1.5.

Because of their foundational nature, there has been extensive study of how to compute histograms privately in non-federerated (i.e., centralized) models. In what follows, we outline a number of different approaches to histogram generation in the federated setting. Table 2 provides a brief qualitative summary of each technique and the main pros and cons.

### 3.1.1 Histograms via Secure Aggregation.

In Section 2.3, we introduced secure aggregation as a primitive to securely compute a sum over data held by $n$ clients. This immediately extends to computing a histogram: each client can form a 'one-hot' vector $x_i$ that encodes their input, so that the $\ell$th entry is 1 (and all other entries are zero) if the client holds the value indexed by $\ell$. Using secure aggregation, we can obtain *the sum of all these sparse vectors*, which is exactly the desired histogram.

As remarked earlier, secure aggregation should be robust to the possibility of clients dropping out during the protocol, while also limiting the amount of communication. We now outline a recent secure aggregation protocol that achieves this while only requiring a single server to perform the aggregation [Bell et al., 2020]. In this protocol, each client $i$ will send messages to the server and a number of other participating clients, so that there is information that can be used to recover from a client dropout. Importantly, we will have to ensure that the extra information does not allow an adversary to recover any client's information.

In this protocol, each client $i$ picks $k$ other clients as "neigbors" at random as the set $N_i$. Each pair of neighbors $i$ and $j$ will agree to use 'masks' $m_{i,j}$ and $m_{j,i}$ to hide their message, with the property that $m_{i,j} = -m_{j,i}$. Each client also distributes arithmetic shares of a randomly chosen value $r_i$ among its set of neighbors. Having done this, client $i$ will send the message $(N_i, x_i + r_i + \sum_{j \in N_i} m_{i,j})$.

The server receives all these messages, and attempts to use them to reconstruct $\sum_{i=1}^{n} x_i$ by interacting with the clients. If all clients stay present for the duration of the computation, then the server can sum the responses from the clients, to obtain $\sum_{i=1}^{n}(x_i + r_i + \sum_{j \in N_j} m_{i,j}) = \sum_i (x_i + r_i)$, since each pair of masks will cancel out in this sum. In this case, the server can go to each neighbor of $i$ and request the share of $r_i$ held by that neighbor, to reconstruct $r_i$ and subtract it from the sum.

However, if a client $i$ drops out early on in the protocol, then the client's input $x_i$ and random value $r_i$ does not reach the server (and is excluded from the computation), but the masks $m_{j,i}$ from the neighbors of $i$ may remain in the sum. In this case, the server can ask each neighbor of $i$ to provide the masks $m_{j,i}$, which can then be deducted from the sum.

In both cases, the server can obtain enough information to calculate the correct sum. We can also convince ourselves that there is no way that an adversarial server could exploit this protocol to unmask a target input $x_i$. To do so, the server would have to learn both the shares of $r_i$ and the values of $m_{j,i}$ from the neighbor set of $i$. However, in the protocol, the server is only allowed to ask each client for at most one of these values, and so there is insufficient information for the server to single out any client's input.

There is still a risk if some clients collude with the server to unmask a target client's value. To handle this case, we can analyze the protocol under the assumption that there are a bounded number of such "corrupt" clients. The full protocol also relaxes the constraints slightly by adopting threshold secret sharing. Here, the secret value is still split into $k$ different pieces, but now only $t$ out of $k$ are required in order to reconstruct the value. This can be achieved by representing the secret as a degree-$t$ polynomial $f()$, and create $k$ different evaluations of $f$ at distinct points.

Given $t$-out-of-$k$ threshold sharing, it is possible to show that the above protocol succeeds (does not allow any node to be attacked) provided that each of the following is true:

- Fewer than $t$ neighbors of each node are corrupt.
- The graph of neighbors remains connected after removing corrupt or dropped-out clients.
- Each node has at least $t$ neighbors surviving after dropouts.

The analysis of the protocol [Bell et al., 2020] shows that each of these conditions holds with high probability provided we choose $k$ to be large enough, i.e., logarithmic in $n$, the total number of clients. In this case, the amount of communication is bounded: each client only has to send their masked message to the server, plus their list of neighbors, and send a constant amount of information to each neighbor. The total cost is then $O(d \log n)$ communication per client to build the histogram.

This approach can be extended to handle additional constraints. In the case that the domain size $d$ is very large, it seems wasteful to send messages of size $d$ when each client's data is described by $O(\log d)$ bits. Lower communication is possible (in the multiple server model) by taking advantage of "distributed point functions" (DPFs) which can more compactly encode sparse vectors [Gilboa and Ishai, 2014]. We might also want some assurance that clients are following the protocol correctly, and only sending a unit weight vote, rather than trying to vote multiple times for one option, or sending negative votes for a disfavored option. This can be achieved by adopting "zero knowledge proofs" (also in the multiple server setting) to ensure that each client's vote is well-formed [Corrigan-Gibbs and Boneh, 2017].

| **Technique** | **Procedure** | **Variance** |
|---|---|---|
| Randomized response ($d = 1$) | Flip one bit | $O(1/(\exp(\epsilon) - 1)^2)$ |
| Unary encoding | Flip bits of a bitstring | $O(\exp(\epsilon)/(\exp(\epsilon) - 1)^2)$ |
| Direct Encoding | Pick an index | $O((\exp(\epsilon + d - 2)/\exp(\epsilon - 1))^2)$ |
| Optimal hashing | Hash then encode | $O(\exp(\epsilon)/(\exp(\epsilon) - 1)^2)$ |
| Sketching | Sketch then encode | $O((\exp(\epsilon) + 1)^2/(\exp(\epsilon) - 1)^2)$ |

Table 3: Techniques for frequency oracles in LDP with variance as a function of privacy parameter $\epsilon$ and dimension $d$.

In summary, the secure aggregation approach to building histograms inherits the *security guarantees* of secure aggregation, and similarly does not protect any leakage from the output — there is *no inbuilt privacy*.

### 3.1.2 LDP Histograms via frequency oracles.

In the Local Differential Privacy literature, the histogram problem is often referred to as constructing a 'frequency oracle', and has been the subject of extensive study [Wang et al., 2017]. Different approaches to *perturbing user inputs* are relevant depending on the size of the domain, $d$.

- For the binary domain ($d = 2$), the simple application of randomized response described in Section 2.5 can be applied. This is the optimal choice (in terms of minimizing communication cost and privacy/accuracy tradeoff).
- For small domains, we can apply some generalization of randomized response. One generalization is to take the $d$-bit one-hot vector representing the client's input, and apply binary randomized response to each bit [Erlingsson et al., 2014]. Alternatively, we can apply a 'direct encoding' approach. This picks the true label with probability $p$, and picks each other label with probability $(1 - p)/(d - 1)$. Applying the $\epsilon$-differential privacy constraint leads to setting $p = \exp(\epsilon)/(\exp(\epsilon) + d - 1)$.
- For larger domains (say, $d$ bigger than 10), better results are obtained by an approach based on hashing [Wang et al., 2017]. That is, each client picks a random hash function that maps the set of inputs from $[d] \to [g]$, for some small $g \ll d$. Typically, $g = 2$ or $g = 3$ is used. Then direct encoding is applied to the hashed value from the client, who reports the perturbed output and the hash function to the server.
- For very large domains ($d$ at least thousands in size), more advanced techniques are needed to cope with the scale of the data. Dimensionality reduction techniques (also called 'sketching') are used to map the very large domain $d$ down to a smaller space, while preserving the sparsity of the input. This is then treated as the input to the histogram, and techniques from above (direct encoding, hashing etc.) are applied to report the results. This sketching approach is at the heart of the deployed techniques for LDP aggregation from Google and Apple [Erlingsson et al., 2014, Differential Privacy Team at Apple, 2017, Bassily et al., 2020].

We summarize the different LDP mechanisms for histogram construction in Table 3, and provide the variance for each as a function of the differential privacy parameter $\epsilon$, and the size of the domain, $d$. Smaller variance is preferable due to higher accuracy, but other factors include the size of the message and the time for the server to reconstruct the histogram. The values of the variance are quoted from Cormode et al. [2021]. These LDP approaches share the same pros and cons as other LDP algorithms: they achieve strong *privacy guarantees*, but can only provide relatively *poor utility*.

### 3.1.3 Distributed Noise Generation for histograms.

If we can generate distributed noise that can be summed to give differentially private noise, then it is quite direct to use this for histogram collection. We can represent each user's input as a one-hot vector, and add independent distributed noise to each entry 2.5.3. The *summation of all these noised vectors* will be the exact histogram, plus DP noise, yielding central differential privacy guarantees. As mentioned in Section 2.5.3, it is desirable that the noise be discrete, for compatibility with secure aggregation methods. For the guarantees to hold, we need to ensure that *enough clients participate* so that the right privacy level is met.

The most well-known DP noise distributions, Laplace and Gaussian noise, are continuous, and hence not well suited to this setting. However, many distributions are known for this task which are discrete:

- Binomial noise can be generated by each client adding Bernoulli noise to the input — i.e., noise that is either zero or one. For $n$ large enough, the Binomial distribution converges to the Gaussian distribution, so offers similar privacy guarantees. This intuition is formalized by Dwork *et al.* [Dwork et al., 2006], and tightened in subsequent work [Ghazi et al., 2019].
- Summing values from the Polya distribution obtains the Geometric distribution, also known as the discrete Laplace distribution, which achieves $(\epsilon, 0)$-differential privacy [Balle et al., 2020]. In practice the noise obtained is almost indistinguishable from the Laplace distribution.
- The skellam mechanism is generated as the different of two Poisson distribution, and is discrete, and closed under summation (inheriting these properties from the Poisson distribution). The sum of skellam distributions is similar to a Gaussian distribution in shape, and provides strong privacy guarantees under composition [Agarwal et al., 2021].
- Other distributed noise distributions include the Arete distribution (tailored to the high privacy setting [Pagh and Stausholm, 2022]), and the Poisson Binomial Mechanism, which reduces the communication cost incurred, particularly in the high privacy case [Chen et al., 2022].

### 3.1.4 Differentially Private histograms via sampling.

Recent work shows how to obtain differentially private histograms in the federated setting by applying a sampling approach [Cormode and Bharadwaj, 2022]. The idea is that the randomness of sampling can introduce a sufficient level of uncertainty to meet the DP requirements, provided we apply some additional contraints to the output. The method, referred to as "sample and threshold" has two main steps:

- The sampling step performs *sampling over a set of $n$ clients*: each client is independently included in the sample with probability $p$. The reports from these clients are collected, and the true histogram of their values is computed.
- The thresholding step then applies a threshold $\tau$ to the histogram: any counts below $\tau$ are removed from the histogram. The resulting histogram is then published.

By judicious choice of the parameters $p$ and $\tau$, the resulting output histogram can be shown to be $(\epsilon, \delta)$ differentially private (where the private parameters are a function of $p$ and $\tau$). For instance, choosing $p = 0.1$ and $\tau = 20$ achieves $\epsilon = 1, \delta = 10^{-8}$ differential privacy.

To operationalize this mechanism, it is necessary to perform the sample-and-threshold steps so that the clients which are sampled are not disclosed. This can be done in various ways. For instance, a trusted entity (secure enclave) could be employed to contact the clients. The thresholding step could be performed in the secure multiparty model [Davidson et al., 2022]. The model may be best suited to scenarios where some amount of sampling is inherent, i.e., when the clients who participate in the federated computation can be considered to be a random sample from a much larger population of eligible candidates. Consequently, the approach may be relatively *simple to implement*, but may need security or *trust assumptions* to ensure that the sampled clients are kept secret.

### 3.1.5 Histogram Applications.

Given a primitive to compute histograms over federated data, it can be applied to the data at different levels of granularity in order to solve a variety of other problems. These are typically cases where the input domain, $\mathcal{D}$ can be mapped to the integers $1 \ldots d$, or some other ordered domain. Histograms can then be applied to coarsened versions of the domain, e.g., by splitting the domain into four non-overlapping quarters. Some example problems include:

- **Heavy hitters.** The heavy hitters problem is to find those items that have a very high count, when the domain size $d$ is very large. There is a natural hierarchical approach: we build a histogram of a coarsened version of the domain (say, broken in to the first half and second half), and then recursively explore those parts which are associated with a high count. Sub-ranges of the domain with low counts can be passed over, since they cannot contain any heavy hitter. By progressively refining the regions of interest, the search can find the heavy hitters while only working with histograms of moderate size. If the regions are at least halved each time, then the number of rounds of this search procedure is bounded by $O(\log d)$.

- **Prefix queries and range queries.** A range query asks for the sum of the number of items that fall in the range $[l, r]$ for parameters $l$ and $r$, while a prefix query is a range query where $l = 1$. Both can be answered by posing a custom histogram query where clients map their inputs into values that correspond to whether the input falls into the range of interest or not. However, we can use a fixed set of histogram queries to collect data that can be used to answer all possible range queries. Specifically, we can collect one histogram on the item sets $\{1\}, \{2\}, \ldots \{d\}$ (the regular histogram), one on the pairs $\{1, 2\}, \{3, 4\}, \{5, 6\} \ldots$, a histogram on groups of four $\{1, 2, 3, 4\}, \{5, 6, 7, 8\}$, and so on. From these $\log_2 d$ histograms, we can write any prefix or any range as the combination of $O(\log d)$ entries, and sum the corresponding entries. Bounding the number of entries to sum for this 'dyadic range' approach is important, since it limits the amount of noise accrued when working with differentially private histograms [Cormode and Yi, 2020].

- **Quantiles.** The quantiles describe the distribution of the data values: the median is the value such that at most half the inputs are below is and at most half are above it. The quantiles generalize this so that the $q$-quantile has at most a $q$ fraction of the input points below it, and $1 - q$ above it. Given a solution for prefix queries, we can find the $q$-quantile by searching the domain $\mathcal{D}$ for the least value whose prefix query satisfies the $q$-quantile requirement.

- **Marginal queries.** The marginals of a multidimensional distribution describe the distribution projected down to a few dimensions, typically just one or two. Marginal queries are used widely in understanding and describing data, such as in building low-degree graphical models of data to allow inference and prediction. The marginals can be captured via histograms, where each histogram bucket holds all the input items that contribute to one marginal entry. Alternatively, coarser grain marginal distributions can be built by combining results from multiple buckets within finer grain marginal histograms.

## 3.2 Mean estimation

Computing the sum or mean of data values is another foundational operation in data analysis that enables a range of tasks, from simple statistics to more complex training of machine learning models.

### 3.2.1 Scalar Mean estimation.

When each client holds a scalar value, and we wish to compute the population mean, it would appear that this is a relatively simple task: without privacy, we can compute the global sum of values (or the sum of a sample), and scale this by the number of participants to obtain a mean value. However, introducing privacy requirements, along with a desire to minimize communication, adds complexity to this task, and a number of different approaches have been suggested. For simplicity, we describe them under the assumption that the input $x$ is a real value in the range $[0, 1]$.

- **Randomized rounding**. Given input $x$ with $0 \leq x \leq 1$, with probability $x$, output 1 otherwise output 0. The idea is that inputs close to zero are more likely to be mapped to zero, while inputs close to one are more likely to be mapped to one, so that by taking the average of a large enough number of rounded inputs, we should get a good estimate of the average of the original inputs. Mathematically, the expected value of this procedure is $x$, but only 1 bit is required to encode the result. Summing the result of many independently rounded values will lead to an accurate estimate of the sum. To achieve (local) differential privacy, we can apply randomized response to each bit, so that bits are flipped with some small probability. This does not change the fact that we obtain an unbiased estimator, but increases the variance of the process.

- **Subtractive dithering**. A similar seeming, but slightly more accurate, approach is to apply randomization to the test for rounding. Under subtractive dithering, we pick a threshold $t$ uniformly from $[0, 1]$, and report both $t$ and whether $x > t$. The estimate is made by computing $I(x > t) + t - 1/2$, where the $I$ function reports 1 if its argument is true, and 0 otherwise. This report similarly gives an unbiased estimator for $x$. This too can be made private via applying randomized response to the binary report $(x > t)$, and then forming an unbiased estimator from the sum of the noisy reports [Basat et al., 2021].

- **Bit-pushing**. For inputs with fixed precision, we can obtain better accuracy by (adaptively) sampling bits from a binary representation, focusing more effort on bits that contribute most to the result. Since each response is just a single bit, then randomized response can again be used for privacy. The estimate of the mean is found by summing up the estimates of each bit in the binary representation, scaled by its weight [Cormode and Markov, 2021].

- **Privacy-aware compression**. Instead of treating privacy as an afterthought, we can define a randomized procedure that handles both rounding and privacy, by designing a probability matrix mapping (discretized) inputs to a smaller range of outputs. The (local) differential privacy propety can be encoded as a set of constraints, then a loss function can be optimized to minimize the variance [Chaudhuri et al., 2022].

Each of these approaches achieves a different trade-off between accuracy, privacy, and the communication cost for different input data distributions. Empirically, subtractive dithering works well when the mean is reasonably large (close to 1), while adaptive approaches such as bit-pushing can handle cases where the mean falls in a narrow but unknown range [Cormode and Markov, 2021].

### 3.2.2 Vector mean estimation.

Vector mean estimation is heavily depended on due to its importance in federated learning: core protocols such as FedSGD and FedAvg rely on computing the (weighted) mean of a collection of vectors, corresponding to gradient updates [McMahan et al., 2017]. Many subsequent variations have been proposed to improve the communication-accuracy-convergence tradeoff, but most ultimately rely on some form of vector averaging. This averaging motivates the design of secure aggregation protocols.

- The essence of the **FedSGD** algorithm is to follow the usual stochastic gradient descent (SGD) algorithm, but where the data remains on client devices. In SGD, a "batch" of labeled training examples is fed into the current model, and the prediction of the model is compared to the known labels. This is used to create a gradient vector on the model weights, to improve the accuracy of the model. This gradient vector is formed as the sum of the gradients owing to each example. In the federated case, the server sends the current model to a subset of clients who each hold some examples. Hence, the batch becomes the set of examples held by the selected clients. Each client locally computes the gradient vector for its clients, and sends these to the server. The server sums the received vectors, and uses these to update the model for the next round. This is repeated many times with different subsets of clients, to obtain the final model.

- The **FedAVG** algorithm stems from the observation that FedSGD cleaves too closely to the central SGD algorithm, and does not take advantage of the fact that each client can do more work. FedAVG shares the same outline, but rather than performing a single step of SGD, the clients perform multiple steps. That is, they compute the gradient against the current model, but then apply this update to their local copy of the model, and iterate the process for some number of steps. Now the clients send the parameters of their partially-trained model (instead of proposed updates to the proposed model), and the server creates a new model by averaging these model weights. It is not immediately obvious that this averaging of model weights is a meaningful step, but there is strong empirical and theoretical evidence that this leads to good results.

Note that these two methods operate in different ways: FedSGD takes the average of model weight updates, while FedAvg take the scaled sum of model weights. However, the underlying computational operation is the same: to find a sum or mean of vectors (which may be subsequently scaled). Hence, we seek federated approaches to finding the vector mean.

A simple approach to vector mean estimation is to apply a scalar mean estimation algorithm on each component of the vector, such as those described in Section 3.2.1. However, there are further opportunities for improvement. The main focus in federated vector mean estimation is to obtain an accurate estimate, while achieving some privacy guarantee, and minimizing the amount of communication. Communication is often reduced by applying a quantization technique: reducing the level of detail used to describe the values. For instance, instead of using 64-bit variables to describe model weights, it is common to reduce these to 16-bit or 8-bit variables, or even single bits [Reisizadeh et al., 2020, Stock et al., 2020].

Further, it may be possible to use correlations in the vector weights to further reduce the communication needed. In the simplest case, the vectors may be very sparse, having many entries that are zero or close to zero. Compact encodings are possible that will take advantage of this sparsity. More generally, it may be possible to perform some generic transform to reduce the dimensionality of the vector being encoded. Such transformations are often in the form of rotations, and may be chosen at random Suresh et al. [2017], Vargaftik et al. [2021, 2022]. Importantly, these transformations and encodings have to be chosen in conjunction with the privacy and security techniques adopted. For privacy, noise may be added to each coordinate, and clipping applied to bound the magnitude in any dimension (hence reducing the level of noise needed).

Many variations of these two approaches to FL are possible, based on the hyperparameters of the SGD algorithm. For instance, the gradient update is scaled by the learning rate parameter, which affects how quickly the process converges. Most other hyperparameters from the centralized case (e.g., batch size) can similarly be applied in the federated setting. In addition, additional privacy and security can be obtained by using Secure Aggregation to sum up the (discretized) gradients, and by adding differential privacy noise to them.

### 3.3 Federated Algorithms summary

In summary, we have described examples of federated algorithms that trade off accuracy, privacy, security, computation and communication costs. When considering a proposed protocol, we can ask

- Where and what noise is added to give privacy?
- What primitives are used to achieve security (e.g., secure aggregation, or more advanced secure multiparty computation)?
- What accuracy and computational costs can be guaranteed?

We saw examples of techniques that can address histograms, quantiles, heavy hitters and range queries; and means of scalars and vectors, and their generalizations to variance and higher moments. In Section 5, we will discuss more examples of queries that are of interest in the federated setting.

## 4 Practical Considerations

Moving the federated approach from prototype results to practical deployments brings with it many challenges. We recap the main challenges here, and discuss these in more detail in subsequent sections.

- **Technical:** how to perform computations in a distributed and private way?
- **Availability and Latency:** updates from clients may be slow of missing, and clients may not always be available.
- **Robustness:** deployed systems need to be robust to malformed responses and malicious actors.
- **Energy:** consideration has to be given to the energy costs of distributing the computation.
- **Fairness:** we also have to address which groups are the most influential in federated computations, and whether this is fair.
- **Privacy:** what privacy model should be adopted, and how should its parameters be set?
- **Data distribution:** real data is often skewed, sparse, non-uniformly distributed and non-stationary.
- **Device heterogeneity:** user devices can vary widely in capability and functionality.
- **Scale:** the solutions we might consider may vary considerably depending on whether we are handling tens of participants or millions.

## 4.1 Privacy Granularity and Budgeting

There are multiple choices around what granularity to apply the definition of differential privacy:

- **Group level privacy** is when enough noise is added to protect the data contributed by a (fixed-size) group of users.
- **User level privacy** is when the noise is calibrated to mask the entire footprint of a user in the data.
- **Device level privacy** is when the unit of privacy protection is a single device in the data.
- **Event level privacy** is when the noise is chosen to mask a single event in the data.

We could similarly achieve other notions of privacy, e.g., app level privacy, based on the needs of the scenario. All of these are feasible, but entail different levels of noise. Event level privacy is typically the least noise, but may allow inferences to be drawn if an adversary wants to study behaviors that may span across multiple events associated with a user.

Most work on privacy focuses on providing bounds for a single query. If multiple queries are run, or the output of the same query is taken multiple times (a longitudinal query) on the same population, then we need to consider the ensuring that some meaningful overall $\epsilon$-DP guarantee holds over the collection of queries — a process known as 'privacy budgeting'. In this setting we need to ask how to set the overall privacy targets, and whether we can justify "refreshing" the privacy budget on a periodic basis, or considering privacy as a resource with a finite limit.

## 4.2 Availability and latency of clients

**Availability** refers to the fact that not all clients may be able to participate in the federated computation, due to eligibility criteria that are enforced. It is common to require that the client only send data when a mobile device has a wifi connection, and is charging. This is to ensure that the federated computation does not draw down the battery power available. However, the consequence of this is that many devices may be limited to only participating during night-time hours. A challenge for designers of federated systems is to set eligibility criteria so that enough devices can participate in the protocol, without affecting the utility of the device to its owner. A related consideration is to ensure that any logging of data on device, and pre-processing is similarly lightweight and does not overwhelm the device.

In addition, **latency** issues can affect protocols: clients may be slow to respond to a request, or not respond at all. This can occur for simple pragmatic reasons: a device may have lost network connectivity, be switched off, or be logged out. Receiving late responses to a stale request can sometimes still be of value if these responses can be incorporated into the aggregation. If the number of dropouts is very large, the process may be unable to continue. Some simple remedies are to approach additional clients, or to increase the number of requests in advance, in anticipation of some percentage of dropouts. Alternatively, we may try to design asynchronous algorithms that can still proceed even while clients are reporting in late.

## 4.3 Federated Data characteristics

It is often the case that the data observed in practical federated scenarios can be challenging to work with, in various dimensions.

- **Data imbalance:** some clients have many records to contribute to a query, while most others have only a very few. This can reflect, for instance, a few 'power users' who make extensive use of a service, while most make only sporadic use. It requires a decision on how to incorporate this imbalance into a federated computation: should we ensure that all clients contribute an equal weight, and so sample or summarize the power users? Or try to capture the global distribution of records, despite this non-uniform allocation?
- **Non-IID local data distributions:** the distribution of data can vary a lot from client to client. This can happen for instance if different clients are writing text in different languages, or capturing images in different locations (urban vs. rural). What this means is that randomly sampling clients is not the same as randomly sampling records, and we may need to adopt a more extensive or stratified approach to client sampling to ensure that we capture a representative set of records.

- **Skewness:** numeric values in federated settings, e.g., latencies, are often drawn from long-tailed distributions. Some data transformation (e.g., a logarithmic transform) may be needed to use them within methods for federated data collection. Simultaneously, some clipping may be necessary to ensure that the noise for privacy does not grow too large.
- **Non-stationarity:** the distribution of client data often changes over time. The consequence for federated computations is that we should avoid long-lived queries, and look at ways to take this non-stationarity into account (caching local data to capture historical behavior, and avoiding methods that are too sensitivity to data changes).

## 4.4 Building a federated stack

The ultimate goal of working on federated computation should be to develop and deploy a robust federated stack, capable of handling key federated components, such as

- **Register a new task** for eligible clients to participate in. Ideally, this should provide a high-level language for task specification.
- **Allocate tasks to clients** subject to their constraints (desire to participate, available privacy budget remaining). The allocation should balance the distribute of clients given each task.
- **Receive responses** from clients and compute the results for downstream consumption. The stack should be responsible that all privacy guarantees are met before release.
- **Manage access** to the computed statistics and related results, with appropriate security protections around them.

It is natural to conceive of the state as being handled by a single entity, but as systems scale, the stack itself may need to be distributed over multiple machines.

## 4.5 Federation in Practice

Practical federated systems are moving from research prototypes into wide scale production. This is particularly true for federated learning systems. A few representative examples include:

- Google's adoption and promotion of federated computation, with accessible explanations in comic form (see `https://federated.withgoogle.com`).
- Meta's use of federated learning within the Papaya system [Huba et al., 2022], and their associated FLSim tool (see `https://github.com/facebookresearch/FLSim`).
- The Musketeer FL system developed in association with IBM (see `https://musketeer.eu`).

In parallel, the first generation of federated analytics tools, based around the Local Differential Privacy model, have been used to gather statistics:

- Google's RAPPOR system, focused on collecting frequency statistics of web browsing [Erlingsson et al., 2014].
- Apple's DP deployment, gathering frequency statistics on typing patterns [Differential Privacy Team at Apple, 2017].
- Microsoft's telemetry monitoring work, concerned with app usage statistics [Ding et al., 2017].

The use of LDP in these systems has been criticized for offering a poor accuracy-privacy tradeoff. Typical instantiations guarantee only a large value of the privacy parameter, $\epsilon$, typically 8-16, while still requiring many tens of thousands of clients to participate in order to get useful results [Differential Privacy Team at Apple, 2017, Erlingsson et al., 2014]. For the next generation of FA tools, it may be preferable to look to other privacy models, such as the combination of distributed noise generation and secure aggregation or TEEs.

Anecdotally, the following observations hold in practice. Each round of federated computation is relatively fast: typically, completing within a few minutes. This is clearly much slower than a centralized computation, but fast compared to the diurnal cycle of use from a mobile device. It's common for federated learning tasks to operate on batches of a few hundreds of clients, over hundreds or thousands of rounds, while analytics tasks work on batches of thousands to millions of clients, but for only one or a few rounds. In both settings, working with more clients reduces the error due to privacy noise and sampling, and so can reduce the number of rounds needed.

The algorithms used in practice appear to be fairly robust to non-adversarial disruptions. The accuracy is only modestly affected by having clients dropout. However, debugging federated computations is currently hard: the privacy concerns of working with large volumes of real data mean that it is typically not possible to allow data logging for comparison, or re-running a computation with identical settings. Instead, it is necessary to test extensively with realistic data in simulated environments before code is deployed to production environments.

**Example: heat maps.** A case-study is given by Bagdasaryan et al. [2022], who seek a way to generate a heatmap of geospatial activity data. Methods based on local differential privacy are considered and rejected, since they require too high a level of noise. Another challenge is that some geographic regions (say, downtown in a city) are very dense, and merit a fine grain decomposition; meanwhile, some regions (say, unpopulated rural regions) are sparsely populated, and have a much lower signal-to-noise ratio. It is desirable to take an adaptive approach to building out the heat map.

The approach makes use of a number of ideas discussed in this survey. For privacy, the authors propose a distributed approach to noise generation, based on the Polya distribution, combined with secure aggregation. This leads to a total amount of noise comparable to the central DP case, but without the need to trust an aggregator.

To handle sparsity, the authors propose to use adaptive hierarchical histograms. An initial uniform histogram is build over the data with a coarse granularity (few buckets). Then, recursively, this histogram is expanded: regions with low activity are left alone, while regions with high activity are subdivided into finer regions to report more information. Conceptually, this is similar to the trie-based heavy hitters approach discussed above to find frequent strings (Section 3.1.5). The paper emphasises the need for differentially private histograms as a primitive to support the algorithm.

Empirical results show that the adaptive approach is able to reconstruct the geospatial activity distribution with low error, measured via mean squared error. The accuracy is comparable to that of approaches that only measure at the finest granularity, but uses much less communication between server and clients to compute this. It is much more accurate than approaches that use non-adaptive randomized histograms (sketches) to represent the data.

## 5 Advanced Topics and Open Problems

In this section, we visit some of the open problems and questions to inform future work in this area.

### 5.1 Algorithmic Challenges

There are many natural data analytics tasks that need new approaches to provide effective federated algorithms. Many of these are statistical in nature. The task of hypothesis testing requires building appropriate test statistics over the distributed data to compare to threshold values. In addition, it may be necessary to adjust the test in order to take account of privacy noise or sampling bias [Wang et al., 2015]. It is often necessary to compare distributions, to measure the similarity of two sources of data. For instance, we may want to measure how similar a distribution of client values is between this week and last week. This necessitates capturing sufficient information to compute or sketch the difference between the distributions. Lastly, many problems can be described as instances of statistical modeling: we want to build a simple graphical or regression model of data. This can be decomposed into two pieces: structure learning — determining the appropriate correlations between features to capture — and parameter learning — estimating the numerical parameters of the chosen model. In both cases, appropriate privacy protection is needed to ensure that the information does not divulge sensitive information about the participants.

Our discussion so far has focused on data that can naturally be represented as scalars, sets or vectors. However, other data seen in the wild requires more complex representations, with additional semantics. It is natural to seek federated ways to learn this information. For instance, we can think of graph data, where each client holds a collection of edges, and the task is to understand the connectivity and shortest-path behavior in this data. Or, clients may hold weights which collectively build up a graph of interactions, from which we would like to extract a low-rank approximate factorization. Many such problems have been studied in various distributed or private models, but rarely have they combined both these requirements.

## 5.2 Models of federated privacy

The model of federated computation, and associated privacy and security models, can be viewed as a work in progress. Although there is strong interest in the privacy gains to be had from keeping data on device, there is not yet complete consensus on how to obtain the right balance of usability and protections for data. Currently, various combinations of secure multiparty computation and differential privacy are being proposed. Other directions worthy of further research include stronger security models that are additionally robust to adversarial entities: malicious servers that might try to corrupt the output, or clients that might try to cheat to obtain outcomes in their own interests. Such actions by clients are often referred to as "poisoning attacks", where a client manipulates the input to the protocol (data poisoning), or manipulates the message that they send (model poisoning). It is highly desirable to develop protocols which can detect or prevent such poisoning, so that we can still obtain usable results so long as only a small minority of clients are involved. Other directions involve obtaining tighter guarantees of differential privacy, by leveraging alternate approaches to "privacy accounting", such as the notions of Rényi Differential Privacy [Mironov, 2017] and zero concentrated differential privacy [Bun and Steinke, 2016].

A particular challenge for federated computation is to incorporate the fact that data changes over time. This requires answering several conceptual questions:

- **Semantics:** how should we interpret the result of queries on changing data, and hence how should the results be presented?
- **Streaming analytics:** how can a long-running federated query keep its results up to date? What state should an aggregating server maintain, and with what protections?
- **Longitudinal issues:** how should we manage the privacy bounds (budget) if users can participate multiple times in the data collection?
- **Parallelism:** how should tasks be assigned to different clients when their are many concurrent tasks? How can we ensure that there are meaningful overall privacy guarantees, and also ensure that there is a diverse selection of clients included in each task response?
- **Adaptive analytics:** can tasks be modified in response to the current data distribution, so we receive better insight into changing patterns?

Progress on any of these questions could help broaden the appeal and applicability of federated computation considerably.

## 5.3 Federated Tools and Systems

As noted in Section 4.5, currently there are few public examples of deployed federated systems, and similarly few general purpose tools and systems. The exception to this is that there is a good number of systems which perform or simulate federated learning, offering Python or C++ interfaces. However, there are not yet systems which offer a way to perform general purpose federated computation.

Consequently, there are many research opportunities concerned with advancing the state of the art in federated systems. The most obvious need is to develop production grade open source systems that can handle a broad enough class of federated computations. These in turn need work to design and implement suitable high-level query languages capable of specifying the complex tasks involved. Last, we also identify the need for appropriate user interfaces to be designed to allow non-technical users to launch new federated tasks, and to surface the tradeoffs in choosing appropriate parameter settings.

## 5.4 Federated Learning and beyond

Recall that federated learning (FL) is concerned with tasks surrounding building high quality models of distributed data. The canonical approach to training models is based around (stochastic) gradient descent, where we compute gradients for each example visited, and combine these to propose updates to the model, via back-propagation. This forms the basis of federated approaches to training, by selecting a subset of clients to compute their local gradient against a current model, and combining these gradients to update the model, in one iteration. Putting this into practice involves specifying various details, such as how to sample clients, what steps clients perform locally, and determining certain hyper-parameters such as learning rate, momentum, etc.

A recent survey covers the many and various research directions in federated learning [Kairouz et al., 2021]. Some example topics include tackling:

- **Performance:** to improve the convergence-communication tradeoff.
- **Privacy bounds:** giving better results under strict DP guarantees.
- **Distributed privacy:** finding the best combination of distributed noise generation and federated learning.
- **Robustness:** providing robustness to poisoning attacks, membership inference, and model inversion.
- **Asynchronicity:** leveraging asynchronicity to handle delays and different usage patterns.
- **Model compression:** reducing the size of models and updates to models.
- **Heterogeneity:** ensuring that the learning works well despite heterogeneity in the data.
- **Fairness:** ensuing that the resulting model is fair, based on some appropriate measure of fairness.
- **Ease-of-use:** building usable systems and simulators to facilitate the deployment of FL.

In addition, there are many aspects of federated learning to tackle that go beyond the central question of training, and that have been minimally studied in the federated model. A complete end-to-end ML solution has several steps, of which training is only one. These include feature selection and normalization, model calibration and evaluation, model maintenance and health-checking post deployment. These steps all share the same privacy concerns as the training step, and so demand private federated solutions. Example tasks might be to build a lightweight classifier to act as a baseline, and to identify the most relevant features; and to capture the ROC curve of a learned classifier on real data to track its performance. Recent efforts have formalized these aims and provided solutions based on using federated histograms [Cormode and Markov, 2022].

## 6 Concluding Remarks

In this survey, we have presented the notion of federated computation as an emerging model emphasising secure, private, distributed computation. The subarea of federated learning has already received a very significant amount of attention from the research community, and is on the path to being very widely adopted. Other forms of federated computation, such as tasks based around computing statistics and performing analytics, are arguably of equal importance, and are on the cusp of seeing comparable levels of impact and adoption.

The intent of this overview is to demonstrate that there are many opportunities to contribute to this new area of computation, from the theoretical to the applied. Of particular note, there is a rich ground to bring together techniques from multiparty computation, (differential) privacy and distributed computing to develop new algorithms and protocols for important tasks. There are also many obstacles to overcome in order to design and prove robust systems to support a variety of federated computing workloads.

## References

Naman Agarwal, Peter Kairouz, and Ziyu Liu. The skellam mechanism for differentially private federated learning. In *Advances in Neural Information Processing Systems*, pages 5052–5064, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/285baacbdf8fda1de94b19282acd23e2-Abstract.html.

Eugene Bagdasaryan, Peter Kairouz, Stefan Mellem, Adrià Gascón, Kallista A. Bonawitz, Deborah Estrin, and Marco Gruteser. Towards sparse federated analytics: Location heatmaps under distributed differential privacy with secure aggregation. *Proc. Priv. Enhancing Technol.*, 2022(4):162–182, 2022. doi: 10.56553/popets-2022-0104. URL https://doi.org/10.56553/popets-2022-0104.

Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. Private summation in the multi-message shuffle model. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 657–676. ACM, 2020. doi: 10.1145/3372297.3417242. URL `https://doi.org/10.1145/3372297.3417242`.

Ran Ben Basat, Michael Mitzenmacher, and Shay Vargaftik. How to send a real number using a single bit (and some shared randomness). In *International Colloquium on Automata, Languages, and Programming*, volume 198 of *LIPIcs*, pages 25:1–25:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.ICALP.2021.25. URL `https://doi.org/10.4230/LIPIcs.ICALP.2021.25`.

Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Thakurta. Practical locally private heavy hitters. *J. Mach. Learn. Res.*, 21:16:1–16:42, 2020. URL `http://jmlr.org/papers/v21/18-786.html`.

Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269. ACM, 2020. doi: 10.1145/3372297.3417885. URL `https://doi.org/10.1145/3372297.3417885`.

Akash Bharadwaj and Graham Cormode. An introduction to federated computation. In *International Conference on Management of Data*, pages 2448–2451. ACM, 2022. doi: 10.1145/3514221.3522561. URL `https://doi.org/10.1145/3514221.3522561`.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*, page 1175–1191. Association for Computing Machinery, 2017. ISBN 9781450349468. doi: 10.1145/3133956.3133982. URL `https://doi.org/10.1145/3133956.3133982`.

Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography*, volume 9985 of *Lecture Notes in Computer Science*, pages 635–658, 2016. doi: 10.1007/978-3-662-53641-4_24. URL `https://doi.org/10.1007/978-3-662-53641-4_24`.

Kamalika Chaudhuri, Chuan Guo, and Mike Rabbat. Privacy-aware compression for federated data analysis. In *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands*, volume 180 of *Proceedings of Machine Learning Research*, pages 296–306. PMLR, 2022. URL `https://proceedings.mlr.press/v180/chaudhuri22a.html`.

Wei-Ning Chen, Ayfer Özgür, and Peter Kairouz. The poisson binomial mechanism for unbiased federated learning with secure aggregation. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 3490–3506. PMLR, 2022. URL `https://proceedings.mlr.press/v162/chen22s.html`.

Graham Cormode and Akash Bharadwaj. Sample-and-threshold differential privacy: Histograms and applications. In *International Conference on Artificial Intelligence and Statistics, AISTATS*, volume 151 of *Proceedings of Machine Learning Research*, pages 1420–1431. PMLR, 2022. URL `https://proceedings.mlr.press/v151/cormode22a.html`.

Graham Cormode and Igor L. Markov. Bit-efficient numerical aggregation and stronger privacy for trust in federated analytics. *CoRR*, abs/2108.01521, 2021. URL `https://arxiv.org/abs/2108.01521`.

Graham Cormode and Igor L. Markov. Federated calibration and evaluation of binary classifiers. *CoRR*, abs/2210.12526, 2022. doi: 10.48550/arXiv.2210.12526. URL `https://doi.org/10.48550/arXiv.2210.12526`.

Graham Cormode and Ke Yi. *Small Summaries for Big Data*. Cambridge University Press, 2020. doi: 10.1017/9781108769938.

Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang. Privacy at scale: Local differential privacy in practice. Tutorial at SIGMOD and KDD, 2018.

Graham Cormode, Sam Maddock, and Carsten Maple. Frequency estimation under local differential privacy. In *International Conference on Very Large Data Bases (VLDB)*, 2021.

Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Symposium on Networked Systems Design and Implementation, NSDI*, pages 259–282. USENIX Association, 2017. URL `https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs`.

Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC*, 2006.

Alex Davidson, Peter Snyder, E. B. Quirk, Joseph Genereux, Benjamin Livshits, and Hamed Haddadi. STAR: secret sharing for private threshold aggregation reporting. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 697–710. ACM, 2022. doi: 10.1145/3548606.3560631. URL `https://doi.org/10.1145/3548606.3560631`.

Differential Privacy Team at Apple. Learning with privacy at scale. `https://machinelearning.apple.com/research/learning-with-privacy-at-scale`, 2017.

Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems*, pages 3571–3580, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/253614bbac999b38b5b60cae531c4969-Abstract.html`.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2006. doi: 10.1007/11761679_29. URL `https://doi.org/10.1007/11761679_29`.

Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1054–1067. ACM, 2014. doi: 10.1145/2660267.2660348. URL `https://doi.org/10.1145/2660267.2660348`.

David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Found. Trends Priv. Secur.*, 2(2–3):70–246, dec 2018. ISSN 2474-1558. doi: 10.1561/3300000019. URL `https://doi.org/10.1561/3300000019`.

Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. On the power of multiple anonymous messages. *IACR Cryptol. ePrint Arch.*, page 1382, 2019. URL `https://eprint.iacr.org/2019/1382`.

Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT Annual International Conference on the Theory and Applications of Cryptographic Techniques,*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658. Springer, 2014. doi: 10.1007/978-3-642-55220-5_35. URL `https://doi.org/10.1007/978-3-642-55220-5_35`.

Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, Kaikai Wang, Anthony Shoumikhin, Jesik Min, and Mani Malek. PAPAYA: practical, private, and scalable federated learning. In *Machine Learning and Systems (MLSys)*. mlsys.org, 2022. URL `https://proceedings.mlsys.org/paper/2022/hash/f340f1b1f65b6df5b5e3f94d95b11daf-Abstract.html`.

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Found. Trends Mach. Learn.*, 14(1-2):1–210, 2021. doi: 10.1561/2200000083. URL `https://doi.org/10.1561/2200000083`.

Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. $\ell$-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3–es, mar 2007. ISSN 1556-4681. doi: 10.1145/1217299.1217302. URL `https://doi.org/10.1145/1217299.1217302`.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017. URL `http://proceedings.mlr.press/v54/mcmahan17a.html`.

Ilya Mironov. Rényi differential privacy. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE Computer Society, 2017. doi: 10.1109/CSF.2017.11. URL `https://doi.org/10.1109/CSF.2017.11`.

Takashi Nishide and Kazuo Ohta. Constant-round multiparty computation for interval test, equality test, and comparison. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 90-A(5):960–968, 2007.

Rasmus Pagh and Nina Mesing Stausholm. Infinitely divisible noise in the low privacy regime. In *International Conference on Algorithmic Learning Theory*, volume 167 of *Proceedings of Machine Learning Research*, pages 881–909. PMLR, 2022. URL `https://proceedings.mlr.press/v167/pagh22a.html`.

Sean Peisert. Trustworthy scientific computing. *Commun. ACM*, 64(5):18–21, apr 2021. ISSN 0001-0782. doi: 10.1145/3457191. URL `https://doi.org/10.1145/3457191`.

Vasyl Pihur, Aleksandra Korolova, Frederick Liu, Subhash Sankuratripati, Moti Yung, Dachuan Huang, and Ruogu Zeng. Differentially-private "draw and discard" machine learning. *CoRR*, abs/1807.04369, 2018. URL `http://arxiv.org/abs/1807.04369`.

Daniel Ramage and Stefano Mazzocchi. Federated analytics: Collaborative data science without data collection. `https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html`, 2020.

Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 2021–2031. PMLR, 2020. URL `http://proceedings.mlr.press/v108/reisizadeh20a.html`.

Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: large-scale differentially private aggregation without a trusted core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pages 196–210. ACM, 2019. doi: 10.1145/3341301.3359660. URL `https://doi.org/10.1145/3341301.3359660`.

Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information. In *PODS*, 1998.

Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979. ISSN 0001-0782. doi: 10.1145/359168.359176. URL `https://doi.org/10.1145/359168.359176`.

Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=rJehVyrKwH`.

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

Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. DRIVE: one-bit distributed mean estimation. In *Advances in Neural Information Processing Systems*, pages 362–377, 2021.

Shay Vargaftik, Ran Ben Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. EDEN: communication-efficient and robust distributed mean estimation for federated learning. In *International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 21984–22014. PMLR, 2022. URL `https://proceedings.mlr.press/v162/vargaftik22a.html`.

Teng Wang, Xuefeng Zhang, Jingyu Feng, and Xinyu Yang. A comprehensive survey on local differential privacy toward data statistics and analysis in crowdsensing. *CoRR*, abs/2010.05253, 2020. URL `https://arxiv.org/abs/2010.05253`.

Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. Locally differentially private protocols for frequency estimation. In *USENIX Security Symposium*, pages 729–745. USENIX Association, 2017. URL `https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tianhao`.

Yue Wang, Jaewoo Lee, and Daniel Kifer. Differentially private hypothesis testing, revisited. *CoRR*, abs/1511.03376, 2015. URL `http://arxiv.org/abs/1511.03376`.

S. L. Warner. Randomised response: a survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

Mengmeng Yang, Lingjuan Lyu, Jun Zhao, Tianqing Zhu, and Kwok-Yan Lam. Local differential privacy and its applications: A comprehensive survey. *CoRR*, abs/2008.03686, 2020. URL `https://arxiv.org/abs/2008.03686`.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *NeurIPS*, 2019.