Privacy Accounting and Quality Control in the Sage Differentially Private ML Platform

Mathias Lécuyer, Riley Spahn, Kiran Vodrahalli, Roxana Geambasu, and Daniel Hsu
Columbia University

Abstract

Companies increasingly expose machine learning (ML) models trained over sensitive user data to untrusted domains, such as end-user devices and wide-access model stores. This creates a need to control the data's leakage through these models. We present *Sage*, a differentially private (DP) ML platform that bounds the cumulative leakage of training data through models. Sage builds upon the rich literature on DP ML algorithms and contributes pragmatic solutions to two of the most pressing systems challenges of global DP: *running out of privacy budget* and the *privacy-utility tradeoff*. To address the former, we develop *block composition*, a new privacy loss accounting method that leverages the growing database regime of ML workloads to keep training models endlessly on a sensitive data stream while enforcing a global DP guarantee for the stream. To address the latter, we develop *privacy-adaptive training*, a process that trains a model on growing amounts of data and/or with increasing privacy parameters until, with high probability, the model meets developer-configured quality criteria. Sage's methods are designed to integrate with TensorFlow-Extended, Google's open-source ML platform. They illustrate how a systems focus on characteristics of ML workloads enables pragmatic solutions that are not apparent when one focuses on individual algorithms, as most DP ML literature does.

1 Introduction

Machine learning (ML) is changing the origin and makeup of the code driving many of our applications, services, and devices. Traditional code is written by programmers and encodes algorithms that express business logic, plus a bit of configuration data. We keep sensitive data – such as passwords, keys, and user data – out of our code, because we often ship this code to untrusted locations, such as end-user devices and app stores. We mediate accesses to sensitive data with access control. When we do include secrets in code, or when our code is responsible for leaking user information to an unauthorized entity (e.g., through an incorrect access control decision), it is considered a major vulnerability.

With ML, "code" – in the form of ML models – is learned by an ML platform from *a lot of training data*. Learning code enables large-scale personalization, as well as powerful new applications like autonomous cars. Often, the data used for learning comes from users and is of personal nature, including emails, searches, website visits, heartbeats, and driving behavior. And although ML "code" is derived from sensitive data, it is often handled as secret-free code. ML platforms, such as Google's Tensorflow-Extended (TFX), routinely push models trained over sensitive data to servers all around the world [5, 24, 33, 43] and sometimes to end-user devices [51, 55] for faster predictions. Some companies also push feature models – such as user embedding vectors and statistics of user activity – into model stores that are often times widely accessible within the companies [24, 33, 48]. Such exposure would be inconceivable in a traditional application. Think of a word processor: it might push *your* documents to *your* device for faster access, but it would be outrageous if it pushed *your* documents to *my* (and everyone else's) device!

There is perhaps a sense that because ML models aggregate data from multiple users, they "obfuscate" individuals' data and warrant weaker protection than the data itself. However, this perception is succumbing to growing evidence that ML models can leak specifics about individual entries in their training sets. Language models trained over users' emails for auto-complete have been shown to encode not only commonly used phrases but also social security numbers and credit card numbers that users may include in their communications [7]. Prediction APIs have been shown to be enable testing for membership of a particular user or entry within a training set [49]. Finally, it has long been established both theoretically and empirically that access to too many linear statistics from a dataset – as an adversary might have due to periodic releases of models, which often incorporate statistics used for featurization – is *fundamentally non-private* [3, 12, 23, 25].

As companies continue to disseminate many versions of models into untrusted domains, controlling the risk of data exposure becomes critical. We present *Sage*, an ML platform based on TFX that uses differential privacy (DP) [15] to bound the cumulative exposure of individual entries in a company's sensitive data streams through all the models released from those streams. At a high level, DP randomizes a computation over a dataset (e.g. training one model) to bound

SOSP '19, October 27–30, 2019, Huntsville, ON, Canada

ACM ISBN 978-1-4503-6873-5/19/10…$15.00
https://doi.org/10.1145/3341301.3359639

the leakage of individual entries in the dataset through the output of the computation (the model). Each new DP computation increases the bound over data leakage, and can be seen as consuming part of a *privacy budget* that should not be exceeded; Sage makes the process that generates all models and statistics preserve a *global DP guarantee*.

Sage builds upon the rich literature on DP ML *algorithms* (e.g., [2, 34, 57], see §2.3) and contributes pragmatic solutions two of the most pressing *systems challenges* of global DP: (1) running out of privacy budget and (2) the privacy-utility tradeoff. Sage expects to be given training pipelines explicitly programmed to individually satisfy a parameterized DP guarantee. It acts as a new access control layer in TFX that: mediates all accesses to the data by these DP training pipelines; instantiates their DP parameters at runtime; and accounts for the cumulative privacy loss from all pipelines to enforce the global DP guarantee against the stream. At the same time, Sage provides the developers with: control over the quality of the models produced by the DP training pipelines (thereby addressing challenge (2)); and the ability to release models endlessly without running out of privacy budget for the stream (thereby addressing challenge (1)).

The key to addressing both challenges is the realization that ML workloads operate on *growing databases*, a model of interaction that has been explored very little in DP, and only with a purely theoretical and far from practical approach [11]. Most DP literature, largely focused on *individual algorithms*, assumes either static databases (which do not incorporate new data) or online streaming (where computations do not revisit old data). In contrast, *ML workloads* – which consist of many algorithms invoked periodically – operate on *growing databases*. Across invocations of different training algorithms, the workload both incorporates new data and reuses old data, often times adaptively. It is in that *adaptive reuse of old data coupled with new data* that our design of Sage finds the opportunity to address the preceding two challenges in ways that are practical and integrate well with TFX-like platforms.

To address the running out of privacy budget challenge, we develop *block composition*, the first privacy accounting method that both allows efficient training on growing databases and avoids running out of privacy budget as long as the database grows fast enough. Block composition splits the data stream into *blocks*, for example by time (e.g., one day's worth of data goes into one block) or by users (e.g., each user's data goes into one block), depending on the unit of protection (event- or user-level privacy). Block composition lets training pipelines combine available blocks into larger datasets to train models effectively, but accounts for the privacy loss of releasing a model at the level of the specific blocks used to train that model. When the privacy loss for a given block reaches a pre-configured ceiling, the block is *retired* and will not be used again. However, new blocks from the stream arrive with zero privacy loss and can be used to train future models. Thus, as long as the database adds new blocks fast enough relative to the rate at which models arrive, Sage will never run out of privacy budget for the stream. Finally, block composition allows adaptivity in the choice of training computation, privacy parameters, and blocks to execute on, thereby modeling the most comprehensive form of adaptivity in DP literature.

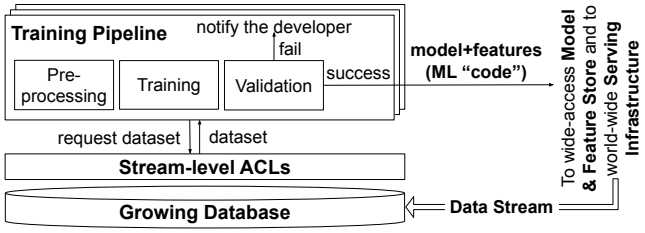


Fig. 1. Typical Architecture of an ML Platform.

To address the privacy-utility tradeoff we develop *privacy-adaptive training*, a training procedure that controls the utility of DP-trained models by repeatedly and adaptively training them on growing data and/or DP parameters available from the stream. Models retrain until, with high probability, they meet programmer-specified quality criteria (e.g. an accuracy target). Privacy-adaptive training uses block composition's support for adaptivity and integrates well with TFX's design, which includes a model validation stage in training pipelines.

2 Background

Our effort builds upon an opportunity we observe in today's companies: the rise of *ML platforms*, trusted infrastructures that provide key services for ML workloads in production, plus strong library support for their development. They can be thought of as *operating systems* for ML workloads. Google has TensorFlow-Extended (TFX) [5]; Facebook has FBLearner [24]; Uber has Michelangelo [33]; and Twitter has DeepBird [32]. The opportunity is to *incorporate DP into these platforms as a new type of access control that constrains data leakage through the models a company disseminates*.

2.1 ML Platforms

Fig. 1 shows our view of an ML platform; it is based on [5, 24, 33]. The platform has several components: *Training Pipelines* (one for each model pushed into production), *Serving Infrastructure*, and a shared data store, which we call the *Growing Database* because it accumulates data from the company's various streams. The access control policies on the Growing Database are exercised through Stream-level ACLs and are typically restrictive for sensitive streams.

The *Training Pipeline* trains a model on data from the Growing Database and verifies that it meets specific quality criteria before it is deployed for serving or shared with other teams. It is launched periodically (e.g., daily) on datasets containing samples from a representative time window (e.g., logs over the past month). It has three customizable modules: (1) *Pre-processing* loads the dataset from the Growing Database, transforms it into a format suitable for training and inference by applying feature transformation operators, and splits the transformed dataset into a training set and a testing

set; (2) *Training* trains the model on a training set; and (3) *Validation* evaluates one or more *quality metrics* – such as accuracy for classification or mean squared error (MSE) for regression – on the testing set. It checks that the metrics reach specific *quality targets* to warrant the model's push into serving. The targets can be fixed by developers or can be values achieved by a previous model. If the model meets all quality criteria, it is bundled with its feature transformation operators (a.k.a. *features*) and pushed into the Serving Infrastructure. The model+features bundle is what we call *ML code*.

The *Serving Infrastructure* manages the online aspects of the model. It distributes the model+features to inference servers around the world and to end-user devices and continuously evaluates and partially updates it on new data. The model+features bundle is also often pushed into a company-wide *Model and Feature Store*, from where other teams within the company can discover it and integrate into their own models. Twitter and Uber report sharing embedding models [48] and tens of thousands of summary statistics [33] across teams through their Feature Stores. To enable such wide sharing, companies sometimes enforce more permissive access control policies on the Model and Feature Store than on the raw data.

2.2 Threat Model

We are concerned with the increase in sensitive data exposure that is caused by applying looser access controls to data-carrying ML code –models+features– than are typically applied to the data. This includes placing models+features in company-wide Model and Feature Stores, where they can be accessed by developers not authorized to access the raw data. It includes pushing models+features, or their predictions, to end-user devices and prediction servers that could be compromised by hackers or oppressive governments. And it includes opening the models+features to the world through prediction APIs that can leak training data if queried sufficiently [49, 53]. Our goal is to "neutralize" the wider exposure of ML codes by making the process of generating them DP across all models+features ever released from a sensitive stream.

We assume the following components are trusted and implemented correctly: Growing Database; Stream-level ACLs; the ML platform code running a Training Pipeline. We also *trust the developer* that instantiates the modules in each pipeline *as long as the developer is authorized* by Stream-level ACLs to access the data stream(s) used by the pipeline. However, we do not trust the wide-access Model and Feature Store or the locations to which the serving infrastructure disseminates the model+features or their predictions. Once a model/feature is pushed to those components, it is considered *released* to the untrusted domain and accessible to adversaries.

We focus on two classes of attacks against models and statistics (see Dwork [22]): (1) *membership inference*, in which the adversary infers whether a particular entry is in the training set based on either white-box or black-box access to the model, features, and/or predictions [3, 23, 25, 49]; and (2) *reconstruction attacks*, in which the adversary infers unknown sensitive attributes about entries in the training set based on similar white-box or black-box access [7, 12, 22].

2.3 Differential Privacy

DP is concerned with whether the output of a computation over a dataset – such as training an ML model – can reveal information about individual entries in the dataset. To prevent such information leakage, *randomness* is introduced into the computation to hide details of individual entries.

Definition 2.1 (Differential Privacy (DP) [20])**.** A randomized algorithm $\mathcal{Q} : \mathcal{D} \to \mathcal{V}$ is (ϵ, δ)-DP if for any $\mathcal{D}, \mathcal{D}'$ with $|D \oplus D'| \leq 1$ and for any $\mathcal{S} \subseteq \mathcal{V}$, we have: $P(\mathcal{Q}(\mathcal{D}) \in \mathcal{S}) \leq e^{\epsilon} P(\mathcal{Q}(\mathcal{D}') \in \mathcal{S}) + \delta$.

The $\epsilon > 0$ and $\delta \in [0, 1]$ parameters quantify the strength of the privacy guarantee: small values imply that one draw from such an algorithm's output gives little information about whether it ran on D or D'. The *privacy budget* ϵ upper bounds an (ϵ, δ)-DP computation's privacy loss with probability $(1-\delta)$. $\oplus$ is a dataset distance (e.g. the symmetric difference [38]). If $|D \oplus D'| \leq 1$, D and D' are *neighboring datasets*.

Multiple mechanisms exist to make a computation DP. They add noise to the computation scaled by its sensitivity s, the maximum change in the computation's output when run on any two neighboring datasets. Adding noise from a Laplace distribution with mean zero and scale $\frac{s}{\epsilon}$ (denoted $laplace(0, \frac{s}{\epsilon})$) gives $(\epsilon, 0)$-DP. Adding noise from a Gaussian distribution scaled by $\frac{s}{\epsilon}\sqrt{2\ln(\frac{1.25}{\delta})}$ gives (ϵ, δ)-DP.

DP is known to address the attacks in our threat model [7, 22, 26, 49]. At a high level, membership and reconstruction attacks work by finding data points that make the observed model more likely: if those points were in the training set, the likelihood of the observed output increases. DP prevents these attacks, as no specific data point can drastically increase the likelihood of the model outputted by the training procedure.

DP literature is very rich and mature, including in ML. DP versions exist for almost every popular ML algorithm, including: stochastic gradient descent (SGD) [2, 57]; various regressions [9, 30, 40, 52, 59]; collaborative filtering [37]; language models [36]; feature selection [10]; model selection [50]; evaluation [6]; and statistics, e.g. contingency tables [4], histograms [56]. The privacy module in TensorFlow v2 implements several SGD-based algorithms [34].

A key strength of DP is its *composition* property, which in its basic form, states that the process of running an (ϵ_1, δ_1)-DP and an (ϵ_2, δ_2)-DP computation on the same dataset is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-DP. Composition enables the development of complex DP computations – such as DP Training Pipelines – from piecemeal DP components, such as DP ML algorithms. Composition also lets one account for (and bound) the privacy loss resulting from a sequence of DP-computed outputs, such as the release of multiple models+features.

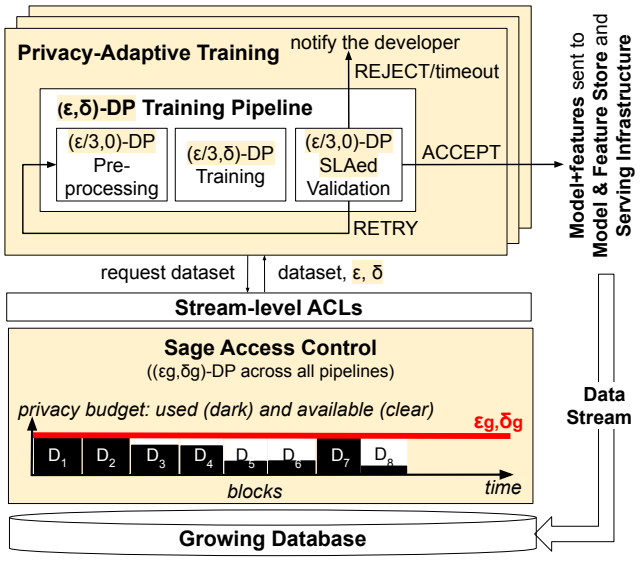


Fig. 2. Sage DP ML Platform. Highlights changes from non-DP version.

A distinction exists between *user-level* and *event-level* privacy. User-level privacy enforces DP on all data points contributed by a user toward a computation. Event-level privacy enforces DP on individual data points (e.g., individual clicks). User-level privacy is more meaningful than event-level privacy, but much more challenging to sustain on streams. Although Sage's design can in theory be applied to user-level privacy (§4.4), we focus most of the paper on *event-level privacy*, which we deem practical enough to be deployed in big companies. §7 discusses the limitations of this semantic.

3 Sage Architecture

The Sage training platform enforces a global (ϵ_g, δ_g)-DP semantic over all models+features that have been, or will ever be, released from each sensitive data stream. The highlighted portions in Fig. 2 show the changes Sage brings to a typical ML platform. First, each Training Pipeline must be made to individually satisfy (ϵ, δ)-DP for some privacy parameters given by Sage at runtime (box (ϵ, δ)*-DP Training Pipeline*, §3.1). The developer is responsible for making this switch to DP, and while research is needed to ease DP programming, this paper leaves that challenge aside.

Second, Sage introduces an additional layer of access control beyond traditional stream-level ACLs (box *Sage Access Control*, §3.2). The new layer splits the data stream into *blocks* and accounts for the privacy loss of releasing a model+features bundle at the level of the specific blocks that were used to train that bundle. In theory, blocks can be defined by any insensitive attribute, with two attributes particularly relevant here: time (e.g., one day's worth of data goes into one block) and user ID (e.g., a user's data goes into the same block). Defining blocks by time provides event-level privacy; defining them by user ID accommodates user-level privacy. Because of our focus on the former, the remainder of this section assumes that blocks are defined by time; §4 discusses sharding by user ID and other attributes.

When the privacy loss for a block reaches the (ϵ_g, δ_g) ceiling, the block is *retired* (blocks D_1, D_2, D_7 are retired in Fig. 2). However, new blocks arrive with a clean budget and can be used to train future models: *as long as the database grows fast enough in new blocks*, the system will never run out of privacy budget for the stream. Perhaps surprisingly, this privacy loss accounting method, which we call *block composition*, is the first practical approach to avoid running out of privacy budget while enabling effective training of ML models on growing databases. §3.2 gives the intuition of block composition while §4 formalizes it and proves it (ϵ_g, δ_g)-DP.

Third, Sage provides developers with control over the quality of models produced by the DP Training Pipelines. Such pipelines can produce less accurate models that fail to meet their quality targets more often than without DP. They can also push in production low-quality models whose validations succeed by mere chance. Both situations lead to operational headaches: the former gives more frequent notifications of failed training, the latter gives dissatisfied users. The issue is often referred to as the *privacy-utility tradeoff* of running under a DP regime. Sage addresses this challenge by wrapping the (ϵ, δ)-DP Training Pipeline into an iterative process that reduces the effects of DP randomness on the quality of the models and the semantic of their validation by invoking training pipelines repeatedly on increasing amounts of data and/or privacy budgets (box *Privacy-Adaptive Training*, §3.3).

3.1 Example (ϵ, δ)-DP Training Pipeline

Sage expects each pipeline submitted by the ML developer to satisfy a parameterized (ϵ, δ)-DP. Acknowledging that DP programming abstractions warrant further research, List. 1 illustrates the changes a developer would have to make at present to convert a non-DP training pipeline written for TFX to a DP training pipeline suitable for Sage. Removed/replaced code is stricken through and the added code is highlighted. The pipeline processes New York City Yellow Cab data [41] and trains a model to predict the duration of a ride.

To integrate with TFX (non-DP version), the developer implements three TFX callbacks. (1) `preprocessing_fn` uses the dataset to compute aggregate features and make user-specified feature transformations. The model has three features: the distance of the ride; the hour of the day; and an aggregate feature representing the average speed of cab rides each hour of the day. (2) `trainer_fn` specifies the model that is to be trained: it configures the columns to be modeled, defines hyperparameters, and specifies the dataset. The model trains with a neural network regressor. (3) `validator_fn` validates the model by comparing test set MSE to a constant.

To integrate with Sage (DP version), the developer: (a) switches library calls to DP versions of the functions (which ideally would be available in the ML platform) and (b) splits the (ϵ, δ) parameters, which are assigned by Sage at runtime,

```
1  def preprocessing_fn(inputs, epsilon):
2    dist_01 = tft.scale_to_0_1(inputs["distance"],0,100)
3    speed_01 = tft.scale_to_0_1(inputs["speed"],0,100)
4    hour_of_day_speed = group_by_meansage.dp_group_by_mean(
5      inputs["hour_of_day"], speed_01, 24, epsilon, 1.0)
6    return {"dist_scaled": dist_01,
7      "hour_of_day": inputs["hour_of_day"],
8      "hour_of_day_speed": hour_of_day_speed,
9      "duration": inputs["duration"]}
10
11 def trainer_fn(hparams, schema, epsilon, delta): [...]
12   feature_columns = [numeric_column("dist_scaled"),
13     numeric_column("hour_of_day_speed"),
14     categorical_column("hour_of_day", num_buckets=24)]
15   estimator = \
16     tf.estimator.DNNRegressorsage.DPDNNRegressor(
17       config=run_config,
18       feature_columns=feature_columns,
19       dnn_hidden_units=hparams.hidden_units,
20       privacy_budget=(epsilon, delta))
21   return tfx.executors.TrainingSpec(estimator,...)
22
23 def validator_fn(epsilon):
24   model_validator = \
25     tfx.components.ModelValidatorsage.DPModelValidator(
26       examples=examples_gen.outputs.output,
27       model=trainer.outputs.output,
28       metric_fn = _MSE_FN, target = _MSE_TARGET,
29       epsilon=epsilon, confidence=0.95, B=1)
30   return model_validator
31
32 def dp_group_by_mean(key_tensor, value_tensor, nkeys,
33   epsilon, value_range):
34   key_tensor = tf.dtypes.cast(key_tensor, tf.int64)
35   ones = tf.fill(tf.shape(key_tensor), 1.0)
36   dp_counts = group_by_sum(key_tensor, ones, nkeys)\
37     + laplace(0.0, 2/epsilon, nkeys)
38   dp_sums = group_by_sum(key_tensor,value_tensor,nkeys)\
39     + laplace(0.0, value_range * 2/epsilon, nkeys)
40   return tf.gather(dp_sums/dp_counts, key_tensor)
```

List. 1. Example Training Pipeline. Shows non-DP TFX (stricken through) and DP Sage (highlighted) versions. TFX API is simplified for exposition.

across the DP function calls. (1) `preprocessing_fn` replaces one call with a DP version that is implemented in Sage: the mean speed per day uses Sage's `dp_group_by_mean`. This function (lines 32-40) computes the number of times each key appears and the sum of the values associated with each key. It makes both DP by adding draws from appropriately-scaled Laplace distributions to each count. Each data point has exactly one key value so the privacy budget usage composes in parallel across keys [38]. The privacy budget is split across the sum and count queries. We envision common functions like this being available in the DP ML platform. (2) `trainer_fn` switches the call to the non-private regressor with the DP implementation, which in Sage is a simple wrapper around TensorFlow's DP SGD-based optimizer. (3) `validator_fn` invokes Sage's DP model validator (§3.3).

3.2 Sage Access Control

Sage uses the composition property of DP to rigorously account for (and bound) the cumulative leakage of data from sensitive user streams across multiple releases of models+features learned from these streams. Specifically, for each sensitive stream, Sage maintains a pre-specified event-level (ϵ_g, δ_g)-DP guarantee across all uses of the stream. Unfortunately, traditional DP composition theory considers either static databases, which leads to wasteful privacy accounting; or purely online streaming, which is inefficient for many ML workloads, including deep neural network training. We thus developed our own composition theory, called *block composition*, which leverages characteristics of ML workloads running on growing databases to permit both efficient privacy accounting and efficient learning. §4 formalizes the new theory. This section describes the limitations of existing DP composition for ML and gives the intuition for block composition and how Sage uses it as a new form of access control in ML platforms.

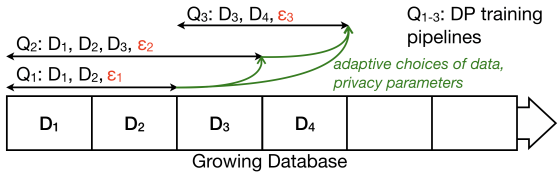


Fig. 3. Characteristics of Data Interaction in ML.

Data Interaction in ML on Growing Databases. Fig. 3 shows an example of a typical workload as seen by an ML platform. Each training pipeline, or *query* in DP parlance, is denoted Q_i. We note two characteristics. First, a typical ML workload consists of multiple training pipelines, training over time on a continuously growing database, and on data subsets of various sizes. For instance, Q_2 may train a large deep neural network requiring massive amounts of data to reach good performances, while Q_3 may train a linear model with smaller data requirements, or even a simple statistic like the mean of a feature over the past day. All pipelines are typically updated or retrained periodically on new data, with old data eventually being deemed irrelevant and ignored.

Second, the data given to a training pipeline – and for a DP model its DP parameters – are typically chosen *adaptively*. For example, the model trained in Q_1 on data from $D_{1,2}$ with budget ϵ_1 may give unsatisfactory performance. After a new block D_3 is collected, a developer may decide to retrain the same model in query Q_2 on data from $D_{1,2,3}$, and with a higher DP budget ϵ_2. Adaptivity can also happen indirectly through the data. Suppose Q_2 successfully trained a recommendation model. Then, future data collected from the users (e.g. in D_4) may depend on the recommendations. Any subsequent query, such as Q_3, is potentially influenced by Q_2's output.

These characteristics imply three requirements for a composition theory suitable for ML. It must support:

- **R1** Queries on overlapping data subsets of diverse sizes.
- **R2** Adaptivity in the choice of: queries, DP parameters, and data subsets the queries process.
- **R3** Endless execution on a growing database.

Limitations of Existing Composition Theory for ML. No previous DP composition theory supports all three requirements. DP has mostly been studied for static databases, where (adaptively chosen) queries are assumed to compute over *the*

entire database. Consequently, composition accounting is typically made at *query level*: each query consumes part of the total available privacy budget for the database. Query-level accounting has carried over even in extensions to DP theory that handle streaming databases [19] and partitioned queries [38]. There are multiple ways to apply query-level accounting to ML, but each trades off at least one of our requirements.

First, one can query overlapping data subsets (**R1**) adaptively across queries, the data used, and the DP parameters [44] (**R2**) by accounting for composition at the query level *against the entire stream*. Queries on Fig. 3 would thus result in a total privacy loss of $\epsilon_1 + \epsilon_2 + \epsilon_3$ over the whole stream. This approach wastes privacy budget and leads to the problem of "running out of budget". Once $\epsilon_g = \epsilon_1 + \epsilon_2 + \epsilon_3$, enforcing a global leakage bound of ϵ_g means that *one must stop using the stream* after query Q_3. This is true even though (1) not all queries run on all the data and (2) there will be new data coming into the system in the future (e.g., D_5). This violates requirement (**R3**) of endless execution on streams.

Second, one can restructure the queries to enable finer granularity with query-level accounting. The data stream is partitioned in blocks, as in Fig. 3. Each query is split into multiple ones, each running with DP on an individual block. The DP results are then aggregated, for instance by averaging model updates as in federated learning [36]. Since each block is a separate dataset, traditional composition can account for privacy loss at the block level. This approach supports adaptivity (**R2**) and execution of the system on streams (**R3**) as new data blocks incur no privacy loss from past queries. However, it violates requirement (**R1**), resulting in unnecessarily noisy learning [13, 14]. Consider computing a feature average. DP requires adding noise once, after summing all values on the combined blocks. But with independent queries over each block, we must add the same amount of noise to the sum over each block, yielding a more noisy total. Additionally, several DP training algorithms [2, 36] fundamentally rely on sampling small training batches from large datasets to amplify privacy, which cannot be done without combining blocks.

Third, one can consume the data stream online using streaming DP. A new data point is allocated to one of the waiting queries, which consumes its entire privacy budget. Because each point is used by one query and discarded, DP holds over the entire stream. New data can be adaptively assigned to any query (**R2**) and arrives with a fresh budget (**R3**). However, queries cannot use past data or overlapping subsets, violating **R1** and rendering the approach impractical for large models.

Block Composition. Our new composition theory meets all three requirements. It splits the data stream into disjoint *blocks* (e.g., one day's worth of data for event-level privacy), forming a growing database on which queries can run on overlapping and adaptively chosen sets of blocks (**R1**, **R2**). This lets pipelines combine blocks with available privacy budgets to assemble large datasets. Despite running overlapping queries, we can still account for the privacy loss of each individual blocks, where each query impacts the blocks it actually uses, *not the entire data stream*. Unused blocks, including future ones, incur no privacy loss. In Fig. 3, the first three blocks each incur a privacy loss of $\epsilon_1 + \epsilon_2$ while the last block has $\epsilon_2 + \epsilon_3$. The privacy loss of these three queries over the entire data stream will only be the maximum of these two values. Moreover, when the database grows (e.g. block D_5 arrives), the new blocks' privacy loss is zero. The system can thus run endlessly by training new models on new data (**R3**).

Sage Access Control. With block composition, Sage controls data leakage from a stream by enforcing DP on its blocks. The company configures a desirable (ϵ_g, δ_g) global policy for each sensitive stream. The Sage Access Control component tracks the available privacy budget for each data block. It allows access to a block until it runs out of budget, after which access to the block will never be granted again. When the Sage Iterator (described in §3.3) for a pipeline requests data, Sage Access Control only offers blocks with available privacy budget. The Iterator then determines the (ϵ, δ) privacy parameters it will use for its iteration and informs Sage Access Control, which deducts (ϵ, δ) from the available privacy budgets of those blocks. Finally, the Iterator invokes the developer-supplied DP Training Pipeline, trusting it to enforce the chosen (ϵ, δ) privacy parameters. §4 proves that this access control policy enforces (ϵ_g, δ_g)-DP for the stream.

The preceding operation is a DP-informed retention policy, but one can use block composition to define other access control policies. Suppose the company is willing to assume that its developers (or user devices and prediction servers in distinct geographies) will not collude to violate its customers' privacy. Then the company could enforce a separate (ϵ_g, δ_g) guarantee for each context (developer or geography) by maintaining separate lists of per-block available budgets.

3.3 Privacy-Adaptive Training

Sage's design adds reliability to the DP model training and validation processes, which are rendered imprecise by the DP randomness. We describe two novel techniques: (1) *SLAed validation*, which accounts for the effect of randomness in the validation process to ensure a high-probability guarantee of correctness (akin to a quality service level agreement, or SLA); and (2) *privacy-adaptive training*, which launches the (ϵ, δ)-DP Training Pipeline adaptively on increasing amounts of data from the stream, and/or with increased privacy parameters, until the validation succeeds. Privacy-adaptive training thus leverages the adaptivity properties of block composition to address DP's privacy-utility tradeoff.

SLAed DP Validation. Fig. 2 shows the three possible outcomes of SLAed validation: ACCEPT, REJECT/timeout, and RETRY. If SLAed validation returns ACCEPT, then with high probability (e.g. 95%) the model reached its configured quality targets for prediction on new data from the same distribution. Under certain assumptions, it is also possible to give

```
1  class DPLossValidator(sage.DPModelValidator):
2    def validate(loss_fn, target, epsilon, confidence, B):
3      if _ACCEPT_test(..., epsilon, (1-confidence)/2, B):
4        return ACCEPT
5      if _REJECT_test(..., epsilon, (1-confidence)/2, B):
6        return REJECT
7      return RETRY
8
9    def _ACCEPT_test(test_labels, dp_test_predictions,
10         loss_fn, target, epsilon, eta, B):
11     n_test = dp_test_predictions.size()
12     n_test_dp = n_test + laplace(2/epsilon)
13     n_test_dp_min = n_test_dp - 2*log(3/(2*eta))/epsilon
14     dp_test_loss = clip_by_value(loss_fn(test_labels,
15       dp_test_predictions), 0, B) + laplace(2*B/epsilon)
16     corrected_dp_test_loss = dp_test_loss +
17       2*B*log(3/(2*eta))/epsilon
18     return bernstein_upper_bound(
19       corrected_dp_test_loss / n_test_dp_min,
20       n_test_dp_min, eta/3, B) <= target
21
22   def bernstein_upper_bound(loss, n, eta, B):
23     return loss+sqrt(2*B*loss*log(1/eta)/n)+4*log(1/eta)/n
```

List. 2. Implementation of `sage.DPLossValidator` used in List. 1.

statistical guarantees of correct negative assessment, in which case SLAed validation returns REJECT. We refer the reader to our extended technical report [31] for this discussion. Sage also supports timing out a training procedure if it has run for too long. Finally, if SLAed validation returns RETRY, it signals that more data is needed for an assessment.

We have implemented SLAed validators for three classes of metrics: loss metrics (e.g. MSE, log loss), accuracy, and *absolute errors* of sum-based statistics such as mean and variance. The technical report [31] details the implementations and proves their statistical and DP guarantees. Here, we give the intuition and an example based on loss metrics. All validators follow the same logic. First, we compute a DP version of the test quantity (e.g. MSE) on a testing set. Second, we compute the *worst-case impact of DP noise* on that quantity for a given confidence probability; we call this a *correction for DP impact*. For example, if we add Laplace noise with parameter $\frac{1}{\epsilon}$ to the sum of squared errors on n data points, assuming that the loss is in $[0, 1]$ we know that with probability $(1 - \eta)$ the sum is deflated by less than $-\frac{1}{\epsilon}\ln(\frac{1}{2\eta})$, because a draw from this Laplace distribution has just an η probability to be more negative than this value. Third, we use known statistical concentration inequalities, also made DP and corrected for worst case noise impact, to upper bound with high probability the loss on the entire distribution.

Example: Loss SLAed Validator. A loss function is a measure of erroneous predictions on a dataset (so lower is better). Examples include: mean squared error for regression, log loss for classification, and minus log likelihood for Bayesian generative models. List. 2 shows our loss validator. The validation function consists of two tests: ACCEPT (described here) and REJECT (described in the technical report [31]).

Denote: the DP-trained model f^{dp}; the loss function range $[0, B]$; the target loss τ_{loss}. Lines 11-13 compute a DP estimate of the number of samples in the test set, corrected for the impact of DP noise to be a lower bound on the true value with probability $(1 - \frac{\eta}{3})$. Lines 14-17 compute a DP estimate of the loss sum, corrected for DP impact to be an upper bound on the true value with probability $(1 - \frac{\eta}{3})$. Lines 18-20 ACCEPT the model if the upper bound is at most τ_{loss}. The bounds are based on a standard concentration inequality (specifically, Bernstein's inequality), which holds under very general conditions [47]. We show in [31] that the Loss ACCEPT Test satisfies $(\epsilon, 0)$-DP and enjoys the following guarantee:

Proposition 3.1 (Loss ACCEPT Test)**.** *With probability at least* $(1 - \eta)$*, the* Accept *test returns true only if the expected loss of* f^{dp} *is at most* τ_{loss}*.*

Privacy-Adaptive Training. Sage attempts to improve the quality of the model and its validation by supplying them with more data or privacy budgets so the SLAed validator can either ACCEPT or REJECT the model. Several ways exist to improve a DP model's quality. First, we can increase the dataset's size: at least in theory, it has been proven that one can compensate for the loss in accuracy due to *any* (ϵ, δ)-DP guarantee by increasing the training set size [28]. Second, we can increase the privacy budget (ϵ, δ) to decrease the noise added to the computation: this must be done within the available budgets of the blocks involved in the training and *not too aggressively*, because wasting privacy budget on one pipeline can prevent other pipelines from using those blocks.

Privacy-adaptive training searches for a configuration that can be either ACCEPTed or REJECTed by the SLAed validator. We have investigated several strategies for this search. Those that conserve privacy budget have proven the most efficient. Every time a new block is created, its budget is divided evenly across the ML pipelines currently waiting in the system. Allocated DP budget is reserved for the pipeline that received it, but privacy-adaptive training will not use all of it right away. It will try to ACCEPT using as little of the budget as possible. When a pipeline is ACCEPTed, its remaining budget is reallocated evenly across the models still waiting in Sage.

To conserve privacy budget, each pipeline will first train and test using a small configurable budget (ϵ_0, δ_0), and a minimum window size for the model's training. On RETRY from the validator, the pipeline will be retrained, making sure to double either the privacy budget if enough allocation is available to the Training Pipeline, or the number of samples available to the Training Pipeline by accepting new data from the stream. This doubling of resources ensures that when a model is ACCEPTed, the sum of budgets used by all failed iterations is at most equal to the budget used by the final, accepted iteration. This final budget also overshoots the best possible budget by at most two, since the model with half this final budget had a RETRY. Overall, the resources used by this DP budget search are thus at most four times the budget of the final model. Evaluation §5.4 shows that this conservative strategy improves performance when multiple Training Pipelines contend for the same blocks.

(a) QueryCompose($\mathcal{A}$, b, r, $(\epsilon_i, \delta_i)_{i=1}^r$):

for i in $1, \ldots, r$ **do** ▷ ($\mathcal{A}$ depends on $V_1^b, \ldots, V_{i-1}^b$ in iter. i)
  $\mathcal{A}$ gives neighboring datasets $\mathcal{D}^{i,0}$ & $\mathcal{D}^{i,1}$
  $\mathcal{A}$ gives (ϵ_i, δ_i)-DP Q_i
  $\mathcal{A}$ receives $V_i^b = Q_i(\mathcal{D}^{i,b})$
return $V^b = (V_1^b, \ldots, V_r^b)$

(a) Traditional Query-level Accounting.

(b) BlockCompose($\mathcal{A}$, b, r, $(\epsilon_i, \delta_i)_{i=1}^r$, $(\text{blocks}_i)_{i=1}^r$):

$\mathcal{A}$ gives two neighboring block datasets $\mathcal{D}^0$ and $\mathcal{D}^1$
for i in $1, \ldots, r$ **do** ▷ ($\mathcal{A}$ depends on $V_1^b, \ldots, V_{i-1}^b$ in iter. i)
  $\mathcal{A}$ gives (ϵ_i, δ_i)-DP Q_i
  $\mathcal{A}$ receives $V_i^b = Q_i(\bigcup_{j \in \text{blocks}_i} \mathcal{D}_j^b)$
return $V^b = (V_1^b, \ldots, V_r^b)$

(b) Block Composition for Static Datasets. Change from query-level accounting shown in yellow background.

(c) AdaptiveStreamBlockCompose($\mathcal{A}$, b, r, $\epsilon_g, \delta_g, \mathcal{W}$):

1: $\mathcal{A}$ gives k, the index of the block with the adversarially chosen change
2: **for** i in $1, \ldots, r$ **do** ▷ ($\mathcal{A}$ depends on $V_1^b, \ldots, V_{i-1}^b$ in iter. i)
3: **if** create new block l and $l == k$ **then**
4: $\mathcal{A}$ gives neighboring blocks $\mathcal{D}_k^0$ and $\mathcal{D}_k^1$
5: **else if** create new block l and $l \neq k$ **then**
6: $\mathcal{D}_l^b = \mathcal{D}(\mathcal{W}, V_1^b, \ldots, V_{i-1}^b)$
7: $\mathcal{A}$ gives blocks_i, (ϵ_i, δ_i), and (ϵ_i, δ_i)-DP Q_i
8: **if** $\bigwedge_{j \in \text{blocks}_i} \text{AccessControl}^j_{\epsilon_g, \delta_g}(\epsilon_1^j, \delta_1^j, \ldots, \epsilon_i^j, \delta_i^j, 0, \ldots)$ **then**
9: $\mathcal{A}$ receives $V_i^b = Q_i(\bigcup_{j \in \text{blocks}_i} \mathcal{D}_j^b)$
10: **else** $\mathcal{A}$ receives no-op $V_i^b = \perp$
return $V^b = (V_1^b, \ldots, V_r^b)$

(c) Sage Block Composition. Adds support for streams (yellow lines 1-6) and adaptive choice of blocks, privacy parameters (green lines 7-8).

Fig. 4. Interaction Protocols for Composition Analysis. $\mathcal{A}$ is an algorithm defining the adversary's power; $b \in \{0, 1\}$ denotes two hypotheses the adversary aims to distinguish; r is the number of rounds; $(\epsilon_i, \delta_i)_{i=1}^r$ the DP parameters used at each round; $(\text{blocks}_i)_{i=1}^r$ the blocks used at each round. $\text{AccessControl}^j_{\epsilon_g, \delta_g}$ returns true if running (ϵ_i, δ_i)-DP query Q_i on block j ensures that with probability $\geq (1 - \delta_g)$ the privacy loss for block j is $\leq \epsilon_g$.

4 Block Composition Theory

This section provides the theoretical backing for block composition, which we invent for Sage but which we believe has broader applications (§4.4). To analyze composition, one formalizes permissible interactions with the sensitive data in a *protocol* that facilitates the proof of the DP guarantee. This interaction protocol makes explicit the worst-case decisions that can be made by modeling them through an adversary. In the standard protocol (detailed shortly), an adversary $\mathcal{A}$ picks the neighboring data sets and supplies the DP queries that will compute over one of these data sets; the choice between the two data sets is exogenous to the interaction. To prove that the interaction satisfies DP, one must show that given the results of the protocol, it is impossible to determine with high confidence which of the two neighboring data sets was used.

Fig. 4 describes three different interaction protocols of increasing sophistication. Alg. (4a) is the basic DP composition protocol. Alg. (4b) is a block-level protocol we propose for static databases. Alg. (4c) is the protocol adopted in Sage; it extends Alg. (4b) by allowing a streaming database and adaptive choices of blocks and privacy parameters. Highlighted are changes made to the preceding protocol.

4.1 Traditional Query-Level Accounting

QueryCompose (Alg. (4a)) is the interaction protocol assumed in most analyses of composition of several DP interactions with a database. There are three important characteristics. First, the number of queries r and the DP parameters $(\epsilon_i, \delta_i)_{i=1}^r$ are fixed in advance. However the DP queries Q_i can be chosen adaptively. Second, the adversary adaptively chooses neighboring datasets $\mathcal{D}^{i,0}$ and $\mathcal{D}^{i,1}$ for each query. This flexibility lets the protocol readily support adaptively evolving data (such as with data streams) where future data collected may be impacted by the adversary's change to past data. Third, the adversary receives the results V^b of running the DP queries Q_i on $\mathcal{D}^{i,b}$; here, $b \in \{0, 1\}$ is the exogenous choice of which database to use and is unknown to $\mathcal{A}$. DP is guaranteed if $\mathcal{A}$ cannot confidently learn b given V^b.

A common tool to analyze DP protocols is *privacy loss*:

Definition 4.1 (Privacy Loss)**.** Fix any outcome $v = (v_1, \ldots, v_r)$ and denote $v_{<i} = (v_1, \ldots, v_{i-1})$. The *privacy loss* of an algorithm Compose($\mathcal{A}$, b, r, $\cdot$) is:

$$\text{Loss}(v) = \ln\left(\frac{P(V^0 = v)}{P(V^1 = v)}\right) = \ln\left(\prod_{i=1}^{r} \frac{P(V_i^0 = v_i | v_{<i})}{P(V_i^1 = v_i | v_{<i})}\right)$$

Bounding the privacy loss for any adversary $\mathcal{A}$ with high probability implies DP [29]. Suppose that for any $\mathcal{A}$, with probability $\geq (1 - \delta)$ over draws from $v \sim V^0$, we have: $|\text{Loss}(v)| \leq \epsilon$. Then Compose($\mathcal{A}$, b, r, $\cdot$) is (ϵ, δ)-DP. This way, privacy loss and DP are defined in terms of distinguishing between two hypotheses indexed by $b \in \{0, 1\}$.

Previous composition theorems (e.g. basic composition [17], strong composition [21], and variations thereof [27]) analyze Alg. (4a) to derive various arithmetics for computing the overall DP semantic of interactions adhering to that protocol. In particular, the basic composition theorem [17] proves that QueryCompose($\mathcal{A}$, b, r, $(\epsilon_i, \delta_i)_{i=1}^r$) is $(\sum_{i=1}^r \epsilon_i, \sum_{i=1}^r \delta_i)$-DP. These theorems form the basis of most ML DP work. However, because composition is accounted for at the query level, imposing a fixed global privacy budget means that one will "run out" of it and stop training models even on new data.

4.2 Block Composition for Static Datasets

Block composition improves privacy accounting for workloads where interaction consists of queries that run on *overlapping data subsets of diverse sizes*. This is one of the characteristics we posit for ML workloads (requirement **R1** in §3.2). Alg. (4b), *BlockCompose*, formalizes this type of interaction for a *static dataset setting* as a springboard to formalizing the full ML interaction. We make two changes to QueryCompose.

First (line 1), the neighboring datasets are defined once and for all before interacting. This way, training pipelines accessing non-overlapping parts of the dataset cannot all be impacted by one entry's change. Second (line 4), the data is split in blocks, and each DP query runs on a subset of the blocks.

We prove that the privacy loss over the entire dataset is the same as the maximum privacy loss on each block, accounting only for queries using this block:

Theorem 4.2 (Reduction to Block-level Composition)**.** *The privacy loss of BlockCompose($\mathcal{A}$, b, r, $(\epsilon_i, \delta_i)_{i=1}^r$, $(\text{blocks}_i)_{i=1}^r$) is upper-bounded by the maximum privacy loss for any block:*

$$|\operatorname{Loss}(v)| \leq \max_k \Big|\ln\Big(\prod_{\substack{i=1\\ k\in\text{blocks}_i}}^{r} \frac{P(V_i^0 = v_i|v_{<i})}{P(V_i^1 = v_i|v_{<i})}\Big)\Big|.$$

Proof. Let $\mathcal{D}^0$ and $\mathcal{D}^1$ be the neighboring datasets picked by adversary $\mathcal{A}$, and let k be the block index s.t. $\mathcal{D}_l^0 = \mathcal{D}_l^1$ for all $l \neq k$, and $|\mathcal{D}_k^0 \oplus \mathcal{D}_k^1| \leq 1$. For any result v of Alg. (4b):

$$\begin{aligned}
\big|\operatorname{Loss}(v)\big| &= \Big|\ln\Big(\prod_{i=1}^{r} \frac{P(V_i^0 = v_i|v_{<i})}{P(V_i^1 = v_i|v_{<i})}\Big)\Big| \\
&= \Big|\ln\Big(\prod_{\substack{i=1\\ k\in\text{blocks}_i}}^{r} \frac{P(V_i^0 = v_i|v_{<i})}{P(V_i^1 = v_i|v_{<i})}\Big) + \cancel{\ln\Big(\prod_{\substack{i=1\\ k\notin\text{blocks}_i}}^{r} \frac{P(V_i^0 = v_i|v_{<i})}{P(V_i^1 = v_i|v_{<i})}\Big)}\Big| \\
&\leq \max_k \Big|\ln\Big(\prod_{\substack{i=1\\ k\in\text{blocks}_i}}^{r} \frac{P(V_i^0 = v_i|v_{<i})}{P(V_i^1 = v_i|v_{<i})}\Big)\Big|
\end{aligned}$$

The slashed term is zero because if $k \notin \text{blocks}_i$, then $\bigcup_{j\in\text{blocks}_i} \mathcal{D}_j^0 = \bigcup_{j\in\text{blocks}_i} \mathcal{D}_j^1$, hence $\frac{P(V_i^0=v_i|v_{<i})}{P(V_i^1=v_i|v_{<i})} = 1$. □

Hence, unused data blocks allow training of other (adaptively chosen) ML models, and exhausting the DP budget of a block means we retire that block of data, and not the entire data set. This result, which can be extended to strong composition (see tech. report [31]), can be used to do tighter accounting than query-level accounting when the workload consists of queries on overlapping sets of data blocks (requirement **R1**). However, it does not support adaptivity in block choice or a streaming setting, violating **R2** and **R3**.

4.3 Sage Block Composition

Alg. (4c), *AdaptiveStreamBlockCompose*, addresses the preceding limitations with two changes. First, supporting streams requires that datasets not be fixed before interacting, because future data depends on prior models trained and pushed into production. The highlighted portions of lines 1-10 in Alg. (4c) formalize the dynamic nature of data collection by having new data blocks explicitly depend on previously trained models, which are chosen by the adversary, in addition to other mechanisms of the world $\mathcal{W}$ that are not impacted by the adversary. Fortunately, Theorem 4.2 still applies, because model training can only use blocks that existed at the time of training, which in turn only depend on prior blocks through DP trained models. Therefore, new data blocks can train new ML models, enabling endless operation on streams (**R3**).

Second, interestingly, supporting adaptive choices in the data blocks implies supporting adaptive choices in the queries' DP budgets. For a given block, one can express query i's choice to use block j as using a privacy budget of either (ϵ_i, δ_i) or $(0, 0)$. Lines 7-8 in Alg. (4c) formalize the adaptive choice of both privacy budgets and blocks (requirement **R2**). It does so by leveraging recent work on DP composition under adaptive DP budgets [44]. At each round, $\mathcal{A}$ requests access to a group of blocks blocks_i, on which to run an (ϵ_i, δ_i)-DP query. Sage's Access Control permits the query only if the privacy loss of each block in blocks_i will remain below (ϵ_g, δ_g). Applying our Theorem 4.2 and [44]'s Theorem 3.3, we prove the following result (proof in [31]):

Theorem 4.3 (Composition for Sage Block Composition)**.** *AdaptiveStreamBlockCompose($\mathcal{A}$,b,r,ϵ_g,δ_g,$\mathcal{W}$) is (ϵ_g, δ_g)-DP if for all k, $\text{AccessControl}^k_{\epsilon_g,\delta_g}$ enforces:*

$$\Big(\sum_{\substack{i=1\\ k\in\text{blocks}_i}}^{r} \epsilon_i(v_{<i})\Big) \leq \epsilon_g \text{ and } \Big(\sum_{\substack{i=1\\ k\in\text{blocks}_i}}^{r} \delta_i(v_{<i})\Big) \leq \delta_g.$$

The implication of the preceding theorem is that under the access control scheme described in §3.2, Sage achieves *event-level (ϵ_g, δ_g)-DP over the sensitive data stream.*

4.4 Blocks Defined by User ID and Other Attributes

Block composition theory can be extended to accommodate user-level privacy and other use cases. The theory shows that one can split a static dataset (Theorem 4.2) or a data stream (Theorem 4.3) into disjoint blocks, and run DP queries adaptively on overlapping subsets of the blocks while accounting for privacy at the block level. The theory focused on *time* splits, but the same theorems can be written for splits based on *any attribute whose possible values can be made public*, such as geography, demographics, or user IDs. Consider a workload on a static dataset in which queries combine data from diverse and overlapping subsets of countries, e.g., they compute average salary in each country separately, but also at the level of continents and GDP-based country groups. For such a workload, block composition gives tighter privacy accounting across these queries than traditional composition, though the system will still run out of privacy budget eventually because no new blocks appear in the static database.

As another example, splitting a stream by user ID enables querying or ignoring all observations from a given user, adding support for user-level privacy. Splitting data over user ID requires extra care. If existing user IDs are not knows, each query might select user IDs that do not exist yet, spending their DP budget without adding data. However, making user IDs public can leak information. One approach is to use incrementing user IDs (with this fact public), and periodically run a DP query computing the maximum user ID is use. This would ensure DP, while giving an estimate of the range of user IDs that can be queried. In such a setting, block composition enables fine-grain DP accounting over queries on any subset

Taxi Regression Task		
Pipelines:	**Configuration:**	
Linear Regression (**LR**)	DP Alg.	AdaSSP from [54], (ϵ, δ)-DP
	Config.	Regularization param $\rho : 0.1$
	Budgets	$(\epsilon, \delta) \in \{(1.0, 10^{-6}), (0.05, 10^{-6})\}$
	Targets	MSE $\in [2.4 \times 10^{-3}, 7 \times 10^{-3}]$
Neural Network (**NN**)	DP Alg.	DP SGD from [2], (ϵ, δ)-DP
	Config.	ReLU, 2 hidden layers (5000/100 nodes) Learning rate: 0.01, Epochs: 3 Batch: 1024, Momentum: 0.9
	Budgets	$(\epsilon, \delta) \in \{(1.0, 10^{-6}), (0.5, 10^{-6})\}$
	Targets	MSE $\in [2 \times 10^{-3}, 7 \times 10^{-3}]$
Avg.Speed x3*	Targets	Absolute error $\in \{1, 5, 7.5, 10, 15\}$ km/h
Criteo Classification Task		
Pipelines:	**Configuration:**	
Logistic Regression (**LG**)	DP Alg.	DP SGD from [35], (ϵ, δ)-DP
	Config.	Learning rate: 0.1, Epochs: 3 Batch: 512
	Budgets	$(\epsilon, \delta) \in \{(1.0, 10^{-6}), (0.25, 10^{-6})\}$
	Targets	Accuracy $\in [0.74, 0.78]$
Neural Network (**NN**)	DP Alg.	DP SGD from [35], (ϵ, δ)-DP
	Config.	ReLU, 2 hidden layers (1024/32 nodes) Learning rate: 0.01, Epochs: 5 Batch: 1024
	Budgets	$(\epsilon, \delta) \in \{(1.0, 10^{-6}), (0.25, 10^{-6})\}$
	Targets	Accuracy $\in [0.74, 0.78]$
Counts x26**	Targets	Absolute error $\in \{0.01, 0.05, 0.10\}$

Tab. 1. Experimental Training Pipelines. *Three time granularities: hour of day, day of week, week of month. **Histogram of each categorical feature.

of the users. While our block theory supports this use case, it suffers from a major practical challenge. New blocks are now created only when new users join the system, so new users must be added at a high rate relative to the model release rate to avoid running out of budget. This is unlikely to happen for mature companies, but may be possible for emerging startups or hospitals, where the stream of incoming users/patients may be high enough to sustain modest workloads.

5 Evaluation

We ask four questions: (**Q1**) Does DP impact Training Pipeline reliability? (**Q2**) Does privacy-adaptive training increase DP Training Pipeline reliability? (**Q3**) Does block composition help over traditional composition? (**Q4**) How do ML workloads perform under Sage's (ϵ_g, δ_g)-DP regime?

Methodology. We consider two datasets: 37M-samples from three months of NYC taxi rides [41] and 45M ad impressions from Criteo [1]. On the Taxi dataset we define a regression task to predict the duration of each ride using 61 binary features derived from 10 contextual features. We implement pipelines for a *linear regression (LR)*, a *neural network (NN)*, and three statistics (average speeds at three time granularities). On the Criteo dataset we formulate a binary classification task predicting ad clicks from 13 numeric and 26 categorical features. We implement a *logistic regression (LG)*, a *neural network (NN)*, and histogram pipelines. Tab. 1 shows details.

Training: We make each pipeline DP using known algorithms, shown in Tab. 1. *Validation:* We use the loss, accuracy, and absolute error SLAed validators on the regression, classification, and statistics respectively. *Experiments:* Each model is assigned a quality target from a range of possible values, chosen between the best achievable model, and the performance of a naïve model (predicting the label mean on Taxi, with MSE 0.0069, and the most common label on Criteo, with accuracy 74.3%). Most evaluation uses privacy-adaptive training, so privacy budgets are chosen by Sage, with an upper-bound of $\epsilon = 1$. While no consensus exists on what a reasonable DP budget is, this value is in line with practical prior work [2, 36]. Where DP budgets must be fixed, we use values indicated in Tab. 1 which correspond to a large budget ($\epsilon = 1$), and a small budget that varies across tasks and models. Other defaults: 90%::10% train::test ratio; $\eta = 0.05$; $\delta = 10^{-6}$. *Comparisons:* We compare Sage's performance to existing DP composition approaches described in §3.2. We ignore the first alternative, which violates the endless execution requirement **R3** and cannot support ML workloads. We compare with the second and third alternatives, which we call *query composition* and *streaming composition*, respectively.

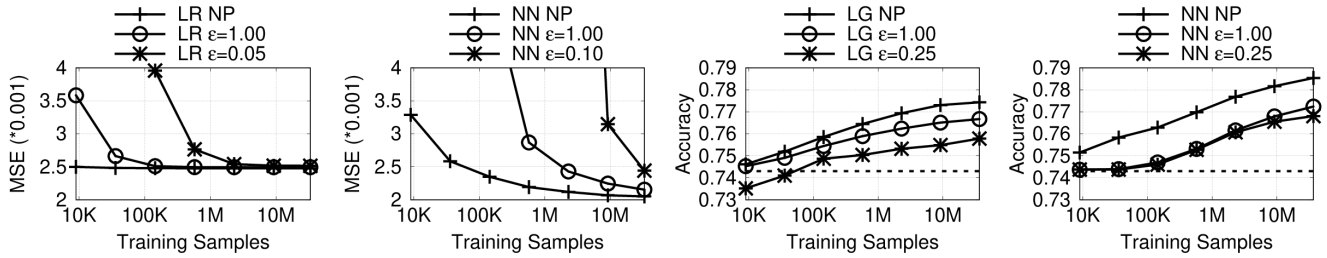
5.1 Unreliability of DP Training Pipelines in TFX (Q1)

We first evaluate DP's impact on model training. Fig. 5 shows the loss or accuracy of each model when trained on increasing amounts data and evaluated on 100K held-out samples from their respective datasets. Three versions are shown for each model: the non-DP version (NP), a large DP budget version ($\epsilon = 1$), and a small DP budget configuration with ϵ values that vary across the model and task. For both tasks, the NN requires the most data but outperforms the linear model in the private and non-private settings. The DP LRs catch up to the non-DP version with the full dataset, but the other models trained with SGD require more data. Thus, model quality is impacted by DP but the impact diminishes with more training data. This motivates privacy-adaptive training.

To evaluate DP's impact on validation, we train and validate our models for both tasks, with and without DP. We use TFX's vanilla validators, which compare the model's performance on a test set to the quality metric (MSE for taxi, accuracy for Criteo). We then re-evaluate the models' quality metrics on a separate, 100K-sample held-out set and measure the fraction of models accepted by TFX that violate their targets on the re-evaluation set. With non-DP pipelines (non-DP training and validation), the false acceptance rate is 5.1% and 8.2% for the Taxi and Criteo tasks respectively. With DP pipelines (DP training, DP validation), false acceptance rates hike to 37.9% and 25.2%, motivating SLAed validation.

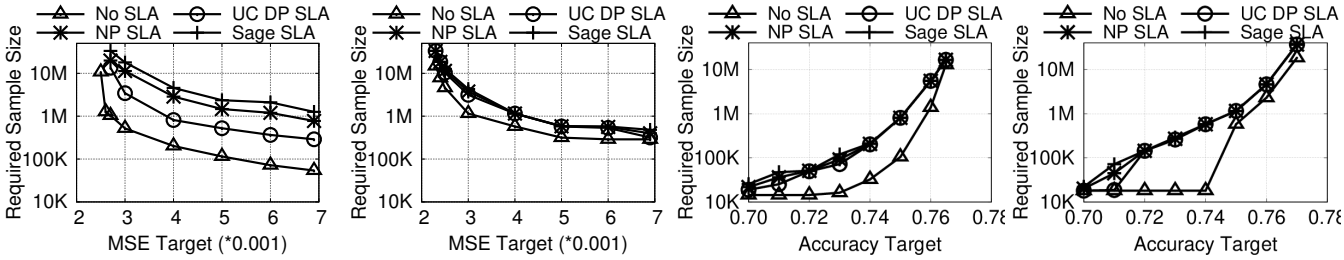
5.2 Reliability of DP Training Pipelines in Sage (Q2)

Sage's privacy-adaptive training and SLAed validation are designed to add reliability to DP model training and validation. However, they may come at a cost of increased data requirements over a non-DP test. We evaluate reliability and sample complexity for the SLAed validation ACCEPT test.



(a) Taxi LR MSE **(b) Taxi NN MSE** **(c) Criteo LG Accuracy** **(d) Criteo NN Accuracy**

Fig. 5. Impacts on TFX Training Pipelines. Impact of DP on the overall performance of training pipelines. 5a, and 5b show the MSE loss on the Taxi regression task (lower is better). 5c; 5d show the accuracy on the Criteo classification task (higher is better). The dotted lines are naïve model performance.



(a) Taxi LR ACCEPT **(b) Taxi NN ACCEPT** **(c) Criteo LG ACCEPT** **(d) Criteo NN ACCEPT**

Fig. 6. Number of Samples Required to ACCEPT models at achievable quality targets. For MSE targets (Taxi regression 6a, and 6b) small targets are harder to achieve and require more samples. For accuracy targets (Criteo classification 6c, and 6d) high targets are harder and require more samples.

Dataset	η	No SLA	NP SLA	UC DP SLA	Sage SLA
Taxi	0.01	0.379	0.0019	0.0172	0.0027
	0.05	0.379	0.0034	0.0224	0.0051
Criteo	0.01	0.2515	0.0052	0.0544	0.0018
	0.05	0.2515	0.0065	0.0556	0.0023

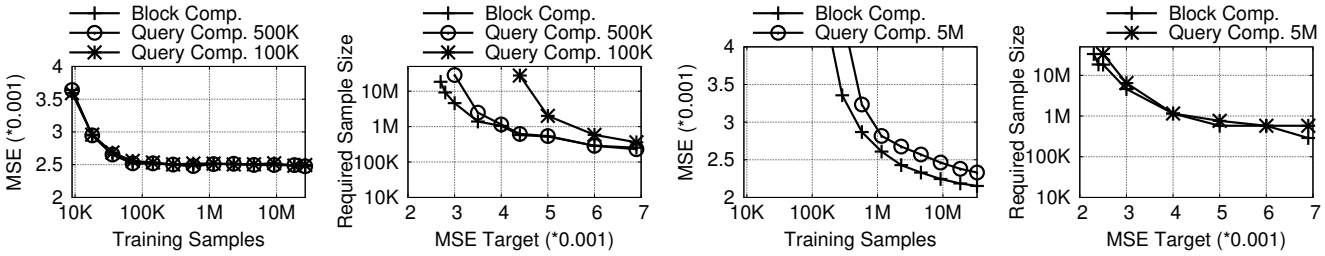
Tab. 2. Target Violation Rate of ACCEPTed Models. Violations are across all models separately trained with privacy-adaptive training.

Tab. 2 shows the fraction of ACCEPTed models that violate their quality targets when re-evaluated on the 100K-sample held-out set. For two confidences η, we show: (1) *No SLA*, the vanilla TFX validation with no statistical rigor, but where a model's quality is computed with DP. (2) *NP SLA*, a non-DP but statistically rigorous validation. This is the best we can achieve with statistical confidence. (3) *UC DP SLA*, a DP SLAed validation without the correction for DP impact. (4) *Sage SLA*, our DP SLAed validator, with correction. We make three observations. First, the NP SLA violation rates are much lower than the configured η values because we use conservative statistical tests. Second, Sage's DP-corrected validation accepts models with violation rates close to the NP SLA. Slightly higher for the loss SLA and slightly lower for the accuracy SLA, but *well below the configured error rates*. Third, removing the correction increases the violation rate by 5x for the loss SLA and 20x for the accuracy SLA, violating the confidence thresholds in both cases, at least for low η. These results confirm that Sage's SLAed validation is reliable, and that correction for DP is critical to this reliability.

The increased reliability of SLAed validation comes at a cost: SLAed validation requires more data compared to a non-DP test. This new data is supplemented by Sage's privacy-adaptive training. Fig. 6a and 6b show the amount of train+test data required to ACCEPT a model under various loss targets for the Taxi regression task. Fig. 6c and 6d show the same for accuracy targets for the Criteo classification task. We make three observations. First, unsurprisingly, non-rigorous validation (No SLA) requires the least data but has a high failure rate because it erroneously accepts models on small sample sizes. Second, the best model accepted by Sage's SLA validation are close to the best model accepted by No SLA. We observe the largest difference in Taxi LR where No SLA accepts MSE targets of 0.0025 while the Sage SLA accepts as low as 0.0027. The best achievable model is slightly impacted by DP, although more data is required. Third, adding a statistical guarantee but no privacy to the validation (NP SLA) already substantially increases sample complexity. Adding DP to the statistical guarantee and applying the DP correction incurs limited additional overhead. The distinction between Sage and NP SLA is barely visible for all but the Taxi LR. For Taxi LR, adding DP accounts for half of the increase over No SLA requiring twice as much data (one data growth step in privacy-adaptive training). Thus, privacy-adaptive training increases reliability of DP training pipelines for reasonable increase in sample complexity.

5.3 Benefit of Block Composition (Q3)

Block composition lets us combine multiple blocks into a dataset, such that each DP query runs over all used blocks with only one noise draw. Without block composition a DP query is split into multiple queries, each operating on a single block, and receiving independent noise. The combined results are more noisy. Fig. 7a and 7c show the model quality of the LR and NN models on the Taxi dataset, when operating on



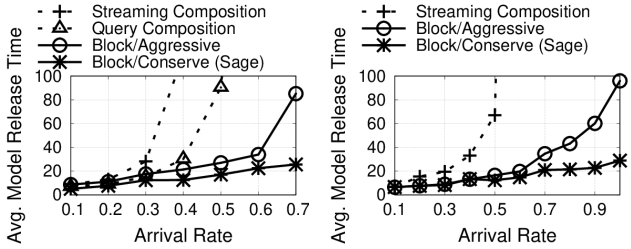
(a) Taxi LR MSE **(b) Taxi LR ACCEPT** **(c) Taxi DNN MSE** **(d) Taxi DNN ACCEPT**

Fig. 7. Block-level vs. Query-level Accounting. Block-level query accounting provides benefits to model quality and validation.

blocks of different sizes, 100K and 500K for the LR, and 5M for the NN. Fig. 7b and 7d show the SLAed validation sample complexity of the same models. We compare these configurations against Sage's combined-block training that allows ML training and validation to operate on their full relevance windows. We can see that block composition helps both the training and validation stages. While LR training (Fig. 7a) performs nearly identically for Sage and block sizes of 100K or 500K (6h of data to a bit more than a day), validation is significantly impacted. The LR cannot be validated with any MSE better than 0.003 with block sizes of 500K, and 0.0044 for blocks of size 100K. Additionally, those targets that can be validated require significantly more data without Sage's block composition: 10x for blocks of size 500K, and almost 100x for blocks of 100K. The NN is more affected at training time. With blocks smaller than 1M points, it cannot even be trained. Even with an enormous block size of 5M, more than ten days of data (Fig. 7c), the partitioned model performs 8% worse than with Sage's block composition. Although on such large blocks validation itself is not much affected, the worse performance means that models can be validated up to an MSE target of 0.0025 (against Sage's 0.0023), and requires twice as much data as with block composition.

5.4 Multi-pipeline Workload Performance (Q4)

Last is an end-to-end evaluation of Sage with a workload consisting of a data stream and ML pipelines arriving over discrete time steps. At each step, a number of new data points corresponding approximately to 1 hour of data arrives (16K for Taxi, 267K for Criteo). The time between new pipelines is drawn from a Gamma distribution. When a new pipeline arrives, its sample complexity (number of data points required to reach the target) is drawn from a power law distribution, and a pipeline with the relevant sample complexity is chosen uniformly among our configurations and targets (Tab. 1). Under this workload, we compare the average model release in steady state for four different strategies. This first two leverage *Query Composition* and *Streaming Composition* from prior work, as explained in methodology and § 3.3. The other two take advantage of Sage's Block Composition. Both strategies uniformly divide the privacy budget of new blocks among all incomplete pipelines, but differ in how each pipeline uses its budget. *Block/Aggressive* uses as much privacy budget as is



(a) Taxi Dataset **(b) Criteo Dataset**

Fig. 8. Average Model Release Time Under Load.

available when a pipeline is invoked. *Block/Conserve (Sage)* uses the Privacy-Adaptive Training strategy defined in § 3.3.

Fig. 8 shows each strategy's average model release time under increasing load (higher model arrival rate), as the system enforces $(\epsilon_g, \delta_g) = (1.0, 10^{-6})$-DP over the entire stream. We make two observations. First, Sage's block composition is crucial. Query Composition and Streaming Composition quickly degrade to off-the-charts release times: supporting more than one model every two hours is not possible and yields release times above three days. On the other hand, strategies leveraging Sage's block composition both provide lower release times, and can support up to 0.7 model arrivals per hour (more than 15 new models per day) and release them within a day. Second, we observe consistently lower release times under the privacy budget conserving strategy. At higher rates, such as 0.7 new models per hour, the difference starts to grow: Block/Conserve has a release time 4x and 2x smaller than Block/Aggressive on Taxi (Fig. 8a) and Criteo (Fig. 8b) respectively. Privacy budget conservation reduces the amount of budget consumed by an individual pipeline, thus allowing new pipelines to use the remaining budget when they arrive.

6 Related Work

Sage's main contribution – block composition – is related to *DP composition theory*. Basic [17] and strong [21, 27] composition theorems give the DP guarantee for multiple queries with adaptively chosen computation. McSherry [38] and Zhang, et.al. [58] show that non-adaptive queries over non-overlapping subsets of data can share the DP budget. Rogers, et.al. [45] analyze composition under adaptive DP parameters, which is crucial to our block composition. These works all consider fixed datasets and query-level accounting.

Compared to all these works, our main contribution is to formalize the new block-level DP interaction model, which supports ML workloads on growing databases while enforcing a global DP semantic without running out of budget. This model sits between traditional DP interaction with static data, and streaming DP working only on current data. In proving our interaction model DP we leverage prior theoretical results and analysis methods. However, the most comprehensive prior interaction model [45] did not support all our requirements, such as interactions with adaptively chosen data subsets, or future data being impacted by previous queries.

Streaming DP [8, 16, 18, 19] extends DP to data streams but is restrictive for ML. Data is consumed once and assumed to never be used again. This enables stronger guarantees, as data need not even be kept internally. However, training ML models often requires multiple passes over the data.

Cummings, et.al. [11] consider *DP over growing databases*. They focus on theoretical analysis and study two setups. In the fist setup, they also run DP workloads on exponentially growing data sizes. However, their approach only supports linear queries, with a runtime exponential in the data dimension and hence impractical. In the second setup, they focus on training a single convex ML model and show that it can use new data to keep improving. Supporting ML workloads would require splitting the privacy budget for the whole stream among models, creating a running out of privacy budget challenge.

A few *DP systems* exist, but none focuses on streams or ML. PINQ [38] and its generalization wPINQ [42] give a SQL-like interface to perform DP queries. They introduce the partition operator allowing *parallel composition*, which resembles Sage's block composition. However, this operator only supports non-adaptive parallel computations on non-overlapping partitions, which is insufficient for ML. Airavat [46] provides a MapReduce interface and supports a strong threat model against actively malicious developers. They adopt a perspective similar to ours, integrating DP with access control. GUPT [39] supports automatic privacy budget allocation and lets programmers specify accuracy targets for arbitrary DP programs with a real-valued output; it is hence applicable to computing summary statistics but not to training ML models. All these works focus on static datasets and adopt a generic, query-level accounting approach that applies to any workload. Query-level accounting would force them to run out of privacy budget if unused data were available. Block-level accounting avoids this limitation but applies to workloads with specific data interaction characteristics (§3.2).

7 Summary and Future Work

As companies disseminate ML models trained over sensitive data to untrusted domains, it is crucial to start controlling data leakage through these models. We presented *Sage*, the first ML platform that enforces a global DP guarantee across all models released from sensitive data streams. Its main contributions are its *block-level accounting* that permits endless operation on streams and its *privacy-adaptive training* that lets developers control DP model quality. The key enabler of both techniques is our systems focus on ML training workloads rather than DP ML's typical focus on individual training algorithms. While individual algorithms see either a static dataset or an online training regime, workloads interact with *growing databases*. Across executions of multiple algorithms, new data becomes available (helping to renew privacy budgets and allow endless operation) and old data is reused (allowing training of models on increasingly large datasets to appease the effect of DP noise on model quality).

We believe that this systems perspective on DP ML presents further opportunities worth pursuing in the future. Chief among them is how to allocate data, privacy parameters, and compute resources to conserve privacy budget while training models efficiently to their quality targets. Sage proposes a specific heuristic for allocating the first two resources (§3.3), but leaves unexplored tradeoffs between data and compute resources. To conserve budgets, we use as much data as is available in the database when a model is invoked, with the lowest privacy budget. This gives us the best utilization of the privacy resource. But training on more data consumes more compute resources. Identifying principled approaches to perform these allocations is an open problem.

A key limitation of this work is its focus on event-level privacy, a semantic that is insufficient when groups of correlated observations can reveal sensitive information. The best known example of such correlation happens when a user contributes multiple observations, but other examples include repeated measurements of a phenomenon over time, or users and their friends on a social network. In such cases, observations are all correlated and can reveal sensitive information, such as a user's demographic attributes, despite event-level DP. It should be noted that even in the face of correlated data DP holds for each individual observation: other correlated observations constitute side information, to which DP is known to be resilient. Still, to increase protection, an exciting area of future work is to add support for and evaluate user-level privacy. Our block accounting theory is amenable to this semantic (§4.4), but finding settings where the semantic can be sustained without running out of budget is an open challenge.

8 Acknowledgements

We thank our shepherd, Thomas Ristenpart, and the anonymous reviewers for the valuable comments. This work was funded through NSF CNS-1351089, CNS-1514437, and CCF-1740833, two Sloan Faculty Fellowships, a Microsoft Faculty Fellowship, a Google Ph.D. Fellowship, and funds from Schmidt Futures and Columbia Data Science Institute.

References

[1] Kaggle display advertising challenge dataset. https://www.kaggle.com/c/criteo-display-ad-challenge, 2014.

[2] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proc. of the ACM Conference on Computer and Communications Security*

(CCS), 2016.

[3] M. Backes, P. Berrang, M. Humbert, and P. Manoharan. Membership privacy in microRNA-based studies. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2016.

[4] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, 2007.

[5] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis, S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich. TFX: A Tensorflow-based production-scale machine learning platform. In *Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

[6] K. Boyd, E. Lantz, and D. Page. Differential privacy for classifier evaluation. In *Proc. of the ACM Workshop on Artificial Intelligence and Security*, 2015.

[7] N. Carlini, C. Liu, U. Erlingsson, J. Kos, and D. Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. arXiv:1802.08232, 2018.

[8] T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *ACM Transactions on Information Systems Security*, 2011.

[9] K. Chaudhuri and C. Monteleoni. Privacy-preserving logistic regression. In *Proc. of the Conference on Neural Information Processing Systems (NeurIPS)*, 2008.

[10] K. Chaudhuri, A. D. Sarwate, and K. Sinha. A near-optimal algorithm for differentially-private principal components. *Journal of Machine Learning Research (JMLR)*, 14, 2013.

[11] R. Cummings, S. Krehbiel, K. A. Lai, and U. Tantipongpipat. Differential privacy for growing databases. In *Proc. of the Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[12] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proc. of the International Conference on Principles of Database Systems (PODS)*, 2003.

[13] J. Duchi and R. Rogers. Lower bounds for locally private estimation via communication complexity. *arXiv:1902.00582*, 2019.

[14] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Minimax optimal procedures for locally private estimation. *Journal of the American Statistical Association*, 2018.

[15] C. Dwork. Differential privacy. In *Automata, languages and programming*. 2006.

[16] C. Dwork. Differential privacy in new settings. In *Proc. of the ACM Symposium on Discrete Algorithms (SODA)*, 2010.

[17] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proc. of the Conference on Theory of Cryptography (TCC)*, 2006.

[18] C. Dwork, M. Naor, T. Pitassi, , and S. Yekhanin. Pan-private streaming algorithms. In *Proc. of The Symposium on Innovations in Computer Science*, 2010.

[19] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, 2010.

[20] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 2014.

[21] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010.

[22] C. Dwork, A. Smith, T. Steinke, and J. Ullman. Exposed! A survey of attacks on private data. *Annual Review of Statistics and Its Application*, 2017.

[23] C. Dwork, A. Smith, T. Steinke, J. Ullman, and S. Vadhan. Robust traceability from trace amounts. In *Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015.

[24] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang. Applied machine learning at Facebook: A datacenter infrastructure perspective. In *Proc. of International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[25] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genetics*, 2008.

[26] B. Jayaraman and D. Evans. Evaluating differentially private machine learning in practice. In *Proc. of USENIX Security*, 2019.

[27] P. Kairouz, S. Oh, and P. Viswanath. The composition theorem for differential privacy. In *International Conference on Machine Learning (ICML)*, 2015.

[28] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM Journal on Computing*, 2011.

[29] S. P. Kasiviswanathan and A. Smith. On the'semantics' of differential privacy: A bayesian formulation. *Journal of Privacy and Confidentiality*, 2014.

[30] D. Kifer, A. Smith, and A. Thakurta. Private convex empirical risk minimization and high-dimensional regression. In *Proc. of the ACM Conference on Learning Theory (COLT)*, 2012.

[31] M. Lecuyer, R. Spahn, K. Vodrahalli, R. Geambasu, and D. Hsu. Privacy accounting and quality control in the sage differentially private ml platform. Online Supplements (also available on arXiv), 2019.

[32] N. Leonard and C. M. Halasz. Twitter meets tensorflow. https://blog.twitter.com/engineering/en_us/topics/insights/2018/twittertensorflow.html, 2018.

[33] L. E. Li, E. Chen, J. Hermann, P. Zhang, and L. Wang. Scaling machine learning as a service. In *Proc. of The International Conference on Predictive Applications and APIs*, 2017.

[34] H. B. McMahan and G. Andrew. A general approach to adding differential privacy to iterative training procedures. *arXiv:1812.06210*, 2018.

[35] H. B. McMahan and G. Andrew. A general approach to adding differential privacy to iterative training procedures. *arXiv:1812.06210*, 2018.

[36] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2018.

[37] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the Netflix prize contenders. In *Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

[38] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, 2009.

[39] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. GUPT: Privacy preserving data analysis made easy. In *Proc. of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012.

[40] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *Proc. of IEEE Symposium on Security and Privacy (S&P)*, 2013.

[41] NYC Taxi & Limousine Commission - trip record data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml, 2018.

[42] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis: a platform for differentially-private analysis of weighted datasets. *Proc. of the International Conference on Very Large Data Bases (VLDB)*, 2014.

[43] S. Ravi. On-device machine intelligence. https://ai.googleblog.com/2017/02/on-device-machine-intelligence.html, 2017.

[44] R. Rogers, A. Roth, J. Ullman, and S. Vadhan. Privacy odometers and filters: Pay-as-you-go composition. In *Proc. of the Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[45] R. M. Rogers, A. Roth, J. Ullman, and S. Vadhan. Privacy odometers and filters: Pay-as-you-go composition. In *Proc. of the Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[46] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for MapReduce. In *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[47] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms. Appendix B.* Cambridge University Press, New York, NY, USA, 2014.

[48] D. Shiebler and A. Tayal. Making machine learning easy with embeddings. SysML http://www.sysml.cc/doc/115.pdf, 2010.

[49] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *Proc. of IEEE Symposium on Security and Privacy (S&P)*, 2017.

[50] A. Smith and A. Thakurta. Differentially private model selection via stability arguments and the robustness of lasso. *Journal of Machine Learning Research*, 2013.

[51] G. P. Strimel, K. M. Sathyendra, and S. Peshterliev. Statistical model compression for small-footprint natural language understanding. arXiv:1807.07520, 2018.

[52] K. Talwar, A. Thakurta, and L. Zhang. Nearly-optimal private LASSO. In *Proc. of the Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[53] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *Proc. of USENIX Security*, 2016.

[54] Y.-X. Wang. Revisiting differentially private linear regression: optimal and adaptive prediction & estimation in unbounded domain. *arXiv:1803.02596*, 2018.

[55] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang. Machine learning at Facebook: Understanding inference at the edge. In *Proc. of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2019.

[56] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. In *Proc. of the IEEE International Conference on Data Engineering (ICDE)*, 2012.

[57] L. Yu, L. Liu, C. Pu, M. E. Gursoy, and S. Truex. Differentially private model publishing for deep learning. In *Proc. of IEEE Symposium on Security and Privacy (S&P)*, 2019.

[58] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau. Ektelo: A framework for defining differentially-private computations. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, 2018.

[59] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. Functional mechanism: Regression analysis under differential privacy. In *Proc. of the International Conference on Very Large Data Bases (VLDB)*, 2012.