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

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

Abstract

We present *Sage*, the first ML platform that enforces a global differential privacy (DP) guarantee across all models produced from a sensitive data stream. Sage extends the Tensorflow-Extended ML platform with novel mechanisms and DP theory to address operational challenges that arise from incorporating DP into ML training processes. First, to avoid the typical problem with DP systems of "running out of privacy budget" after a pre-established number of training processes, we develop *block composition*. It is a new DP composition theory that leverages the time-bounded structure of training processes to keep training models endlessly on a sensitive data stream while enforcing event-level DP on the stream. Second, to control the quality of ML models produced by Sage, we develop a novel *iterative training process* that trains a model on increasing amounts of data from a stream until, with high probability, the model meets developer-configured quality criteria.

1 Introduction

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

With ML, "code" is generated from a training algorithm, a bit of configuration, plus *a lot of training data*. Often, the training data comes from users and is personal, including their private emails, searches, website visits, locations, purchases, heartbeats, and driving behavior. Code learned from user data allows personalization and enables powerful new applications like autonomous driving.

Although ML code incorporates sensitive data, we often handle it as we would secret-free code, rather than as we would handle sensitive data. ML platforms, such as Google's Tensorflow-Extended (TFX), Facebook's FBLearner, and Uber's Michelangelo, routinely push models trained over sensitive data to servers all around the world [5, 26, 32, 40] and sometimes to end-user devices [49, 46] for faster predictions. Some companies also report pushing feature models trained over sensitive data – such as user embedding vectors and statistics of user activity – into shared model stores that are widely accessible within the company, even though they may not enable such wide access for the data itself [26, 32, 43]. 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* device!

There is perhaps a sense that, because ML models aggregate data from multiple users, they "obfuscate" individuals' data sufficiently to warrant weaker protection of the models than of the data itself. However, that perception is succumbing to increasing evidence that ML models can leak substantial information about their training sets. Carlini, et.al. [8] showed that language models trained over users' emails leak secrets – such as passwords, social security numbers, and credit card numbers – that users often include in their communications. Shokri, et.al. [44] showed that membership in a training set can be inferred even when the attacker only has access to a model's external predictions. Calandrino, et.al. [7] showed that recommenders leak information across users. 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 ML models, which often include statistics used for featurization – is *fundamentally non-private* [3, 13, 24, 27].

As companies continue to disseminate many versions of ML codes into untrusted domains, it becomes critical to account for and control the data exposure risks imposed by these codes. We present *Sage*, an ML platform based on TFX that leverages *differential privacy* (DP) [16] to control the cumulative leakage of individual entries in a company's sensitive data streams through all the ML models the company releases from those streams. DP randomizes a computation over a dataset to bound the privacy

*This is an abridged version of a paper that is under submission at a conference [31]. We will make the paper available on arxiv.

loss of individual entries in the dataset through its outputs. Sage makes the process of generating models preserve a global DP guarantee across all models that it generates.

Sage introduces an additional layer of access control into ML platforms beyond the traditional access control lists applied to the raw data. The new layer splits the data stream into fixed-timeframe *blocks* and accounts for the privacy loss incurred from releasing a model or statistic at the level of the blocks that were used to train the model or statistic. When the privacy loss for a given block reaches a pre-configured ceiling, the block is *retired* and will not be used again for ML. However, new blocks from the stream arrive with a clean budget and can be used to train future models. Thus, Sage controls a stream's leakage through ML by controlling ML's access to the stream's blocks.

The preceding design for privacy loss accounting raises substantial semantic and operational challenges for ML platforms. First, perhaps surprisingly, the design cannot be supported practically with existing DP composition theory. We present *block composition*, the first DP composition theory that permits computations on overlapping subsets of data blocks while accounting for privacy loss at the level of the specific blocks that are actually used by each computation. Compared to designs based on traditional DP composition, our design based on block composition leads to more efficient privacy accounting and ultimately to the first practical approach to address DP's "running out of privacy budget" problem for ML workloads on continuous data streams.

Second, incorporating DP into the model generation process decreases the reliability of the ML platform and raises operational challenges. The models may require more data to reach the same levels of accuracy as without DP (if they ever reach it). Their evaluation, which must also be a randomized DP process, may succeed by mere chance, pushing sub-par models into production. We describe *iterative training*, a new process that trains and evaluates an ML model or statistic on increasing amounts of data from a stream until, with high probability, the model meets developer-specified quality criteria.

2 Differential Privacy

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

Definition 1 (Differential Privacy (DP) [21]). *A randomized algorithm $\mathcal{M} : \mathcal{D} \to \mathcal{Y}$ is (ϵ, δ)-DP if for any $\mathcal{D}, \mathcal{D}'$ with Hamming distance $|D \oplus D'| \leq 1$ and for any $\mathcal{S} \subseteq \mathcal{Y}$, we have: $P(\mathcal{M}(\mathcal{D}) \in \mathcal{S}) \leq e^{\epsilon} P(\mathcal{M}(\mathcal{D}') \in \mathcal{S}) + \delta$.*

Here, $\epsilon > 0$ and $\delta \in [0, 1]$ are parameters that quantify the strength of the guarantee: for small values, (ϵ, δ)-DP implies that observing one draw from the algorithm's output gives little information about whether it was run on D or D', i.e., with or without any individual entry.

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 triggered by adding or removing a single entry in the dataset. Adding noise from a Laplace distribution with mean zero and scale $\frac{s}{\epsilon}$ (denoted $\text{Laplace}(0.0, \frac{s}{\epsilon})$) 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.

Known to address the threat of data leakage through ML [23, 44], DP has been studied extensively in this domain. DP versions exist for almost every popular ML algorithm, including: stochastic gradient descent (SGD) [1, 51]; various regressions [10, 30, 37, 47, 53]; collaborative filtering [34]; feature selection [11]; model selection [45]; evaluation [6]; and statistics [4, 50].

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 the privacy loss resulting from a sequence of DP-computed outputs, such as periodic release of models.

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). We focus here on *event-level privacy*.

3 Sage DP ML Platform

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. Google has TensorFlow-Extended (TFX) [5]; Facebook has FBLearner [26]; Uber has Michelangelo [32]. Each operates differently, but they all provide services for training and serving models, as well as storage and access control capabilities that constrain access to the raw data. 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.*

3.1 Overview

Fig. 1 shows Sage's architecture through comparison with a typical ML platform akin to TFX. The differences are highlighted in yellow background. After describing the functioning of the non-DP version and the threat inherent in it, we describe Sage's changes and contributions.

Non-DP ML Platform. A typical ML platform has several components: *Training Pipelines* (one for each model pushed into production), *Serving Infrastructure*, and a *Data Store* shared by all training pipelines and the serving infrastructure. The *Data Store* collects data from streams, often in log files. Its access control policies are often restrictive for sensitive streams. They are typically enforced at stream level with access control lists (ACLs), e.g.: only engineers of product X may access X's data stream.

The *Training Pipeline* trains a model on data from the Data Store and verifies that the model 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 Data Store, transforms it into a format suitable for training and inference, and splits it 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 they 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 [43] and tens of thousands of summary statistics [32] across teams through their Feature Stores. To enable such wide sharing, companies typically enforce more permissive access control policies on the Model and Feature Store than on the raw data.

The Threat. 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 they encode. It includes pushing models+features to end-user devices and prediction servers in physical locations that could be compromised by hackers or oppressive governments. And it includes opening the models+features to the world through predictions that over time can leak training data.

Sage DP ML Platform. Sage's goal is to neutralize the wider exposure of models+features by making the process of generating them (ϵ_g, δ_g)-DP across all models+features that will ever be released from a sensitive stream. The highlighted portions in Fig. 1 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*, §3.2). The developer is responsible for making this switch to DP and Sage *trusts* that training pipelines preserve this guarantee.

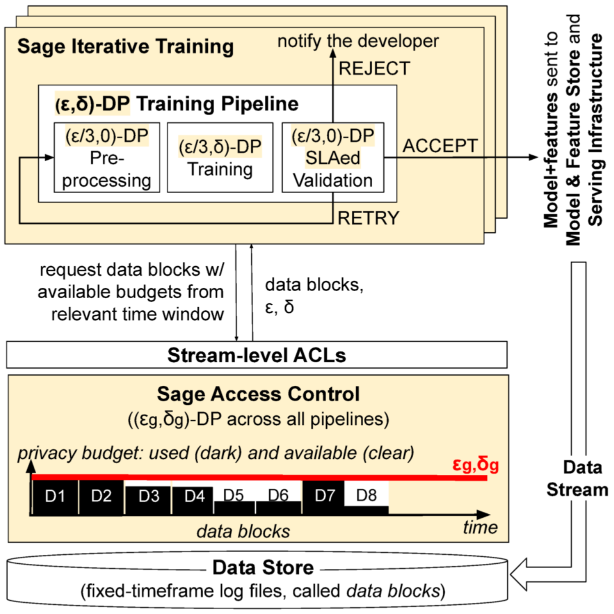

Sage Iterative Training
notify the developer
REJECT
(ε,δ)-DP Training Pipeline
(ε/3,0)-DP Pre-processing
(ε/3,δ)-DP Training
(ε/3,0)-DP SLAed Validation
ACCEPT
RETRY
Model+features sent to Model & Feature Store and Serving Infrastructure
request data blocks w/ available budgets from relevant time window
data blocks, ε, δ
Stream-level ACLs
Sage Access Control
((εg,δg)-DP across all pipelines)
privacy budget: used (dark) and available (clear)
εg,δg
D1
D2
D3
D4
D5
D6
D7
D8
data blocks
time
Data Stream
Data Store
(fixed-timeframe log files, called data blocks)

Fig. 1: **The Sage DP ML Platform.** Changes from a typical non-DP ML platform are highlighted in yellow background.

Second, Sage introduces an additional layer of access control beyond traditional stream-level ACLs (box *Sage Access Control*, §3.3). The new layer splits the data stream into fixed-timeframe *data blocks* (or *blocks*) and accounts for the privacy loss of releasing a model+features at the level of the specific blocks that were used to train the model+features. When the privacy loss for a given block D_i reaches a preconfigured (ϵ_g, δ_g) ceiling, the block is *retired* and will not be used again for training (blocks $D1, D2, D7$ are retired in Fig. 1). However, new blocks from the stream arrive with a clean budget and can be used to train future models. Thus, Sage controls a sensitive data stream's leakage through ML by controlling ML's access to blocks of data from that stream.

Perhaps surprisingly, the preceding design for privacy loss accounting *cannot be supported practically with traditional DP composition*. §3.3 gives the intuition of *block composition*, the first composition theory that permits multiple computations on overlapping subsets of data while accounting for privacy loss at the level of those blocks that are actually used in each computation. The theory, which has significant implications for Sage's practicality, is formalized in our complete paper [31].

```
 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_mean~~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 def trainer_fn(hparams, schema,epsilon, delta): [...]
11   feature_columns = [numeric_column("dist_scaled"),
12     numeric_column("hour_of_day_speed"),
13     categorical_column("hour_of_day", num_buckets=24)]
14   estimator = \
15     ~~tf.estimator.DNNRegressor~~sage.DPDNNRegressor(
16       config=run_config,
17       feature_columns=feature_columns,
18       dnn_hidden_units=hparams.hidden_units,
19       privacy_budget=(epsilon, delta))
20   return tfx.executors.TrainingSpec(estimator,...)
21 def validator_fn(epsilon):
22   model_validator = \
23     ~~tfx.components.ModelValidator~~sage.DPModelValidator(
24       examples=examples_gen.outputs.output,
25       model=trainer.outputs.output,
26       metric_fn = _MSE_FN, target = _MSE_TARGET,
27       epsilon=epsilon, confidence=0.95, B=1)
28   return model_validator
29 def dp_group_by_mean(key_tensor, value_tensor, nkeys,
30     epsilon, value_range):
31   key_tensor = tf.dtypes.cast(key_tensor, tf.int64)
32   ones = tf.fill(tf.shape(key_tensor), 1.0)
33   dp_counts = group_by_sum(key_tensor, ones, nkeys)\
34     + laplace(0.0, 2/epsilon, nkeys)
35   dp_sums = group_by_sum(key_tensor,value_tensor,nkeys)\
36     + laplace(0.0, value_range * 2/epsilon, nkeys)
37   return tf.gather(dp_sums/dp_counts, key_tensor)
```

List. 1: **Example Training Pipeline.** Non-DP TFX (stricken through) and DP Sage (highlighted) versions. TFX API simplified for exposition.

Third, Sage incorporates mechanisms to address some of the operational challenges that DP is bound to raise in practice. Making the Training Pipeline DP introduces new failure modes: it can produce less accurate models that fail to meet their quality targets more often than without DP; and it can push in production low-quality models whose validations succeed by mere chance. Both lead to operational headaches: the former gives more frequent notifications of failed training, the latter gives dissatisfied users. Sage attempts to minimize these operational headaches 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 (box *Sage Iterative Training*, §3.4).

3.2 Example (ϵ, δ)-DP Training Pipeline

Sage expects each pipeline submitted by the Developer to satisfy a parameterized (ϵ, δ)-DP. DP versions exist for virtually any popular ML algorithm, and some have been implemented in ML platforms (e.g., Tensorflow v2 includes DP SGD). As part of Sage, we have implemented DP versions of multiple TFX functions, as needed by our experimental pipelines. We use a simple example to show what programming a DP Training Pipeline would entail with rich library support from the DP training platform.

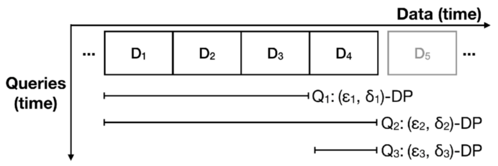


Fig. 2: **Characteristics of Data Interaction in ML.**

List. 1 shows the code changes an ML developer makes to transform a non-DP training pipeline written for TFX to a DP training pipeline suitable for Sage. Changed code is stricken through and replacement code is highlighted. The pipeline processes NYC Yellow Cab data [38] to train a model that predicts 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 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` configures the columns to be modeled, defines hyperparameters, and specifies the training algorithm, here a neural network regressor. (3) `validator_fn` validates the model by comparing testset MSE to baseline, here a constant.

To integrate with Sage (DP version), the developer: (i) switches library calls to DP versions of the functions, which ideally would be available in the ML platform; and (ii) splits the (ϵ, δ) parameters, which are assigned by Sage at runtime, 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 29-37) computes the number of times each key appears and the sum of the values associated with each key. It makes both DP by adding tensors of draws from appropriately-scaled Laplace distributions, one draw for each count and one for each sum. Each data point has exactly one key value so the privacy budget usage composes in parallel across keys [35]. The privacy budget is split across the sum and count queries. We envision common functions like this shipping with 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 optimizer. (3) `validator_fn` invokes Sage's DP validator (§3.4).

3.3 Sage Access Control

Sage uses the composition property of DP to formally account for (and bound) the cumulative leakage of data from sensitive user streams across multiple releases of models+features learned from these streams. For each sensitive stream, Sage maintains a pre-specified event-level (ϵ_g, δ_g)-DP guarantee across every uses of the stream. Unfortunately, directly applying existing DP composition theory leads either to wasteful privacy accounting or to unnecessarily noisy learning. We thus developed our own composition theory, dubbed *block-level privacy accounting*. It leverages characteristics of ML workloads to permit both efficient accounting and efficient learning. Our complete paper formalizes the new theory, which bears broader applications than Sage [31]. This section describes the limitations of existing DP composition for ML workloads and gives the intuition for block-level privacy accounting and how Sage uses it as a new form of access control in ML platforms.

Characteristics of Data Interaction in ML. Fig. 2 shows an example of a typical workload as seen by an ML platform. Each "query," denoted Q_i, corresponds to a training pipeline. We note two characteristics. First, the typical ML workload consists of multiple training pipelines launched at different times and operating on *overlapping windows of potentially very different sizes*. Q_1 may compute a linear regression model, requiring a few days' worth of data for representative results. Q_2 may compute a neural network, requiring many more days of data to fit properly. Q_3 may update a summary statistic computed for each day (such as a feature's mean or variance), requiring only data from the previous day.

Second, training pipelines are typically *adaptive*, i.e. one pipeline may depend either directly or indirectly on previous pipelines. A directly dependent query might be a new query Q_4 that is launched because the model learned by (say) Q_2 does not reach its quality target, so the developer increases the training set size and relaunches the pipeline. The choice of the data on which Q_4 runs, along with its computation and configuration (e.g., model hyperparameter values), is therefore dependent on Q_2's output. An indirectly dependent query would be one whose training is impacted by the output of a previous query *through the collected data*. Suppose Q_2 trains a recommendation model; then, future data collected from the users may depend on its recommendations, so any subsequent query will be 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 both computation and data subsets the queries process.
R3 Endless execution on new data from a stream.

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 available privacy budget for the database. Query-level accounting has carried over even in extensions to DP theory that handle streaming databases [20] and partitioned queries [35]. There are multiple ways to apply query-level accounting to ML, each trading off at least one of the preceding requirements.

First, one can permit queries on overlapping subsets of the data (**R1**) and allow adaptivity across these queries (**R2**) by accounting for composition at query level *against the entire stream.* In Fig. 2, query-level composition gives a total privacy loss of $\epsilon_1 + \epsilon_2 + \epsilon_3$ over the whole stream after running the three queries. This approach wastes privacy budget and leads to the problem of "running out of privacy budget" for the stream. Once $\epsilon_g = \epsilon_1 + \epsilon_2 + \epsilon_3$, enforcing a global leakage bound of ϵ_g means that after executing query Q_3, *one must stop using the stream.* This is true despite the fact that not all queries run on all the data and that there will be new data coming into the system in the future (e.g., block D_5). This behavior violates requirement (**R3**) of endless execution on streams.

Second, one can restructure the training queries to account for privacy loss at a finer granularity with query-level accounting. One partitions the data stream in blocks, as shown on Fig. 2, and splits each training query into multiple queries each running with DP on an individual block. The DP results are then aggregated to give the final answer, for instance by averaging model updates as in federated learning [33]. Since each block is a separate dataset, traditional composition can account for privacy loss at block level. This approach supports query adaptivity (**R2**) and allows new data blocks added to the stream to incur no privacy loss from past queries, thereby enabling endless execution of the system on streams (**R3**). However, by forcing the blocks to not overlap, it violates requirement (**R1**) and results in unnecessarily noisy learning [14, 15]. Consider computing a feature average. DP requires adding noise only 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 of each block, yielding a more noisy total.

Block-level Composition Accounting. We have developed a new composition theory that meets all three requirements. The data stream is also split into non-overlapping, fixed-timeframe *blocks* (e.g., one day's worth of data), but we allow queries to run on overlapping and adaptively chosen sets of blocks (**R1**, **R2**). Despite running overlapping queries, we can still account for the privacy loss at the level of individual blocks, where each query only impacts the blocks it actually uses, *not the entire data stream.* Unused blocks, including future ones, incur no privacy loss. In Fig. 2, the first three blocks each incur a privacy loss of $\epsilon_1 + \epsilon_2$ while the last block has $\epsilon_2 + \epsilon_3$. Thus, after executing these three queries, the privacy loss over the entire data stream will be the maximum of these two values. Moreover, when block D_5 arrives, its privacy loss is zero, so the system can run endlessly by training new models on new data (**R3**).

Sage Access Control. With block-level accounting, Sage controls data leakage from a sensitive stream by controlling DP access to blocks of data from the stream. The company configures a desirable (ϵ_g, δ_g) global policy for each sensitive stream. The Sage Access Control component tracks, for each data block, the available privacy budget for that block. When a new block is added to the Data Store, its available privacy budget is set to (ϵ_g, δ_g). When a block's available privacy budget reaches zero, the block is "retired," i.e., it will never be allowed to be used again in ML computations. When the Sage Iterator (described in §3.4) for a pipeline requests data, Sage Access Control offers only blocks with available privacy budgets. The Iterator then determines the privacy budget (ϵ, δ) it will use for its iteration and informs Sage Access Control, which then deducts (ϵ, δ) from the available privacy budgets of those blocks (assuming they are still available). Finally, the Iterator invokes the developer-supplied DP Training Pipeline, trusting it to enforce the chosen (ϵ, δ) privacy parameters. We prove that this access control policy enforces (ϵ_g, δ_g)-DP across all uses of the sensitive stream [31].

3.4 Sage Iterative 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 give a high-probability guarantee of correctness for the validation (akin to a quality service level agreement, or SLA); and (2) *iterative model training*, which launches the (ϵ, δ)-DP Training Pipeline repeatedly on increasing amounts of data from the stream, and/or with increased privacy parameters, until the validation succeeds.

SLAed DP Validation. Fig. 1 shows the three possible outcomes of SLAed model validation: ACCEPT, REJECT, and RETRY. If SLAed model validation returns ACCEPT, then with high probability (e.g. 95%) the model reaches its configured quality targets when predicting on new data from the same distribution. For certain metrics and under certain assumptions, it is also possible to give statistical guarantees of correct negative assessment. In such cases, if SLAed validation returns REJECT, then with high probability (e.g., 95%) this type of model *will never* reach the configured target, no matter how much data it is trained/-validated on. Finally, if SLAed validation returns RETRY, it signals that it needs more data for an assessment.

We have implemented SLAed validators for three classes of metrics (`sage.DPModelValidator` in List. 1): *loss metrics* (e.g. MSE for regressions), *accuracy metrics* (e.g. for classification), and *absolute errors of sum-based statistics* (e.g. mean, variance). Our complete paper [31] details our implementations and justifies their statistical and DP guarantees. Here, we just include basic intuition. 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.

Iterative Training. Sage attempts to improve the quality of a model and its validation by supplying them with increasing amounts of 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 [29]. 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 affect the ability of other pipelines to train on the same blocks.

Iterative training searches for a configuration that can be either ACCEPTed or REJECTed by the SLAed validator. We have investigated multiple strategies for how to do this search. The ones that have proven most efficient are those that *conserves privacy budget*. Briefly, we start out with a small privacy budget (ϵ_0, δ_0) and with developer-configured start time and minimum window size for the model's training. On RETRY from the validator, we make sure to double either the privacy budget or the number of samples available to the Training Pipeline by accepting new data from the stream. Our evaluation shows that compared to alternatives, the preceding iteration strategy conserves privacy budgets and improves performance when multiple Training Pipelines contend for the same blocks.

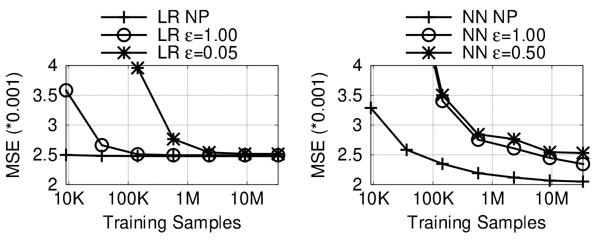
4 Evaluation

We address four questions: (**Q1**) Does DP impact reliability of TFX Training Pipelines? (**Q2**) Does Sage's iterative training with SLAed validation increase reliability of DP Training Pipelines? (**Q3**) Does block-level accounting improve training compared to query-level accounting? (**Q4**) How do workloads of multiple pipelines perform under Sage's (ϵ_g, δ_g)-DP regime?

We develop multiple training pipelines on a public 37M-sample dataset from three months of the NYC taxi dataset [38]. We consider a regression task to predict the duration of each cab ride using 61 binary features derived from 10 contextual features about each cab ride. We im-

Models:	Configuration:	
Linear Regression (**LR**)	DP Alg.	AdaSSP from [48], (ϵ, δ)-DP
	Config.	Regularization param ρ : 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 [1], (ϵ, δ)-DP
	Config.	ReLU, Layers: (5K, 100) 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 $\in [2 \times 10^{-3}, 7 \times 10^{-3}]$
Statistics:	**Configuration:**	
Avg.Speed*	Targets	Absolute error $\in \{1, 5, 7.5, 10, 15\}$ km/h

Tab. 1: **Experimental Training Pipelines.** * Statistics at three time granularities: hour of day, day of week, week of month.



(a) **LR MSE** (b) **NN MSE**

Fig. 3: **Impact of DP on Vanilla Training Pipelines.**

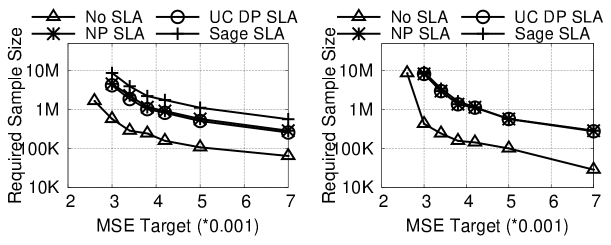
plement two predictive training pipelines – *linear regression (LR)* and *neural network (NN)* – and three summary statistics pipelines – average speeds at three time granularities. Tab. 1 details the configurations and parameters of each training pipeline. *Training:* We make each pipeline DP using known algorithms for the models and the Laplace method for the summary statistics – as indicated in Tab. 1. *Validation:* For the predictive models, we use our loss SLAed validator with MSE as the loss metric. For our task, a naïve model that always returns the average ride duration has an MSE of 0.0069, so that is the upper value we configure for our validation targets. For the summary statistics we use our sum-based SLAed validator, with absolute error as the metric. We instantiate each pipeline for multiple quality targets, shown in Tab. 1, yielding a workload of 25 pipelines. We configure: 90%/10% train/test ratio; $\delta = 10^{-6}$; $\eta = 0.05$.

4.1 Unreliability of TFX DP Training Pipelines (Q1)

Fig. 3 show the loss of LR and NN when trained on increasing amounts of data. Three versions are shown for each model: the non-DP version (NP), a large DP budget version ($\epsilon = 1$), and a small DP budget one ($\epsilon = 0.05$ for LR and $\epsilon = 0.5$ for NN). The NN requires the most data, but achieves the best performance: with sufficient data, the DP NN outperforms the non-DP LR. The DP LRs catch up to the non-DP version with the full dataset, but the DP NNs appear to require more data. Thus, model

η	No SLA	NP SLA	UC DP SLA	Sage SLA
0.01	0.379	0.0019	0.0172	0.0027
0.05	0.379	0.0034	0.0224	0.0051
0.10	0.379	0.0039	0.0240	0.0059

Tab. 2: **Target Violation Rate of ACCEPTed Models.** Violations are across LR and NN separately trained with iterative training.



(a) **LR ACCEPT** (b) **NN ACCEPT**

Fig. 4: **Sample Complexity of SLAed DP Validation.**

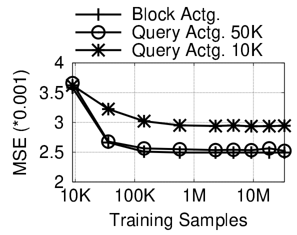
quality is impacted by DP but the impact diminishes with more samples. These results motivate iterative training.

Tab. 2, column No SLA, shows the rate of erroneous acceptance of DP-trained models with higher MSE than their specified targets. Here, validation uses a DP version of the comparison with the target, *without Sage's SLAs*. The false acceptance rate is a little under 40%, a big hike from the 5.1% false rejection rate with no SLA on the non-DP pipeline. This result motivates SLAed validation.

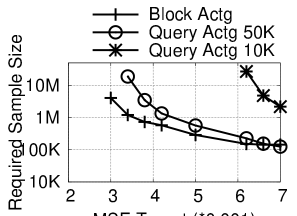
4.2 Reliability of Sage DP Training Pipelines (Q2)

Sage's iterative training and SLAed validators are designed to add reliability to DP model training and validation. Tab. 2 shows the fraction of ACCEPTed models violating their quality targets (based a 100K-sample held-out evaluation set). For three confidences η, we show: (1) *No SLA*, the vanilla TFX validation with no statistical rigor, but where the MSE is DP. (2) *NP SLA*, a non-private but statistically rigorous validation. This is the best we can hope to achieve with statistical confidence. (3) *UC DP SLA*, the DP SLAed validation but without the correction for DP impact. (4) *Sage SLA*, our DP SLAed validator. We make three observations. First, the NP SLA's violation rate is much lower than η: our generalization bounds are conservative for this task. Second, Sage's DP-corrected validation accepts models with violation rates slightly higher than NP SLA, but *well below the configured error rate, η*. Third, removing the correction increases the violation rate by 5x, even violating the confidence threshold for $\eta = 0.01$; this is despite the conservative generalization bounds! These results confirm that (1) Sage's SLAed validation is reliable and (2) its correction for DP impact is critical for reliability.

The increased reliability of SLAed validation comes at a cost: SLAed validation requires more data compared of a non-DP test. This new data is supplemented by Sage's iterative training process. Fig. 4 show the amount of train+test data required to ACCEPT a model under various



(a) **LR MSE**



(b) **Samples for Validation.**

Fig. 5: **Block-level vs. Query-level Accounting.** LR $\epsilon = 1$.

quality targets. First, Sage's SLA validation is able to accept models for targets as low as 0.003, which is close the 0.0025 minimum MSE that the non-rigorous No SLA validation can accept (potentially erroneously). Second, unsurprisingly, non-rigorous validation (No SLA) requires the least data. Adding a statistical guarantee to the validation but no privacy (NP SLA) increases sample complexity substantially. Adding DP to the statistical guarantee and applying the DP correction incur limited additional overhead. The distinction between Sage and NP SLA is barely visible for NN. For LR, it accounts for half of the increase over No SLA as it requires one additional data growth step in iterative training. Thus, SLAed validation and iterative training increase reliability of DP training pipelines for reasonable increase in sample complexity.

4.3 Benefit of Block-level Accounting (Q3)

Block-level privacy accounting lets us combine more or fewer blocks into a dataset, as needed by specific models or statistics, while incurring privacy loss only on blocks used. This effect is also achievable with previous theory by partitioning each DP query into multiple ones operating on a single block and combining results.

Fig. 5 shows (a) model quality and (b) SLAed validation sample complexity of an LR model when operating on blocks of 10K and 50K samples. It compares these configurations against Sage's combined-block training that allows ML training and validation to operate on their full relevance windows. For small block sizes (10K), the partitioned LR model takes a 18% hit in MSE which does not improve with more data (more partitions). The SLAed validation process (b) is even more affected: it is unable to validate models for MSE targets < 0.006! (Recall that just returning the average ride duration has MSE 0.0069 and the best MSE value for non-DP LR is 0.0025.) This is because each single block query receives DP noise: combining blocks increases the data size, but also the amount of noise, which cancels the effect, preventing the model training and the validation test from improving. In contrast, in Sage adding blocks increases the data size but not the noise: we can validate up to MSE targets of 0.003, a significant improvement. For larger blocks (50K), partitioning hurts the model quality less, but still affects SLAed validation.

4.4 Multi-pipeline Workload Performance (Q4)

Our last experiment 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 constant amount of new data arrives and forms a new block. The number of new ML pipelines arriving is drawn from an exponential distribution with arrival rate λ^{-1} (i.e. average number of new pipelines). Each pipeline is then drawn from our 25 configurations of tasks and targets. Pipelines with smaller amounts of required data are more likely to be drawn. For each strategy, we measure the *average release time*, the average number of time steps needed to train+validate a model.

Recall that to discover sufficient resource allocations to meet each ML pipeline's quality target without vastly overspending, Sage iteratively trains and validates models on increasing data and/or privacy budget. We evaluate three iterative training strategies. The first strategy, *Sequential Execution*, is a naïve baseline that can be analyzed with query-level accounting. Models are serialized and allocated data in turn, training on the whole privacy budget for this data until meeting their quality target. Then, the next model trains. Sage's block-level accounting enables training multiple ML pipelines concurrently, on overlapping blocks of data, using subsets of the privacy budget. The other two iterative training strategies take advantage of Sage's block-level accounting, which allows training of multiple ML pipelines concurrently on overlapping blocks of data. Both strategies uniformly divide the privacy budget of new blocks among all incomplete pipelines. They differ in how each pipeline uses the budget. One, *Concurrent, Conservative (Sage)*, uses the strategy defined in § 3.4. The other, *Concurrent, Aggressive*, uses all privacy budget available on each query.

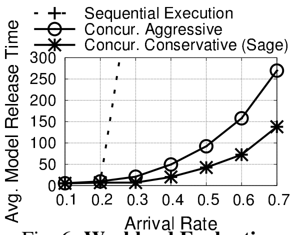


Fig. 6: **Workload Evaluation.**

Fig. 6 shows how the average release times grow for each strategy under increasing load and as the system enforces $(\epsilon_g, \delta_g) = (1.0, 10^{-6})$-DP. We make two observations. First, Sage's block-level accounting and the concurrent training it enables, is crucial: the Sequential Execution produces off-the-charts release times. Second, we observe consistently lower release times under the privacy budget conserving strategy. Privacy budget conservation reduces the amount of privacy budget that will be consumed by an individual pipeline and allows new pipelines to use the remaining budget when they arrive in the system.

5 Related Work

Basic [18] and strong [22, 28] *DP composition* give the DP guarantee for multiple queries with adaptively chosen computation. McSherry [35] and Zhang, et.al. [52] show that non-adaptive queries over non-overlapping subsets of data can share the DP budget. Rogers, et.al. [41] analyze composition under adaptive DP parameters, which is crucial to our block-level accounting. These works all consider fixed datasets and query-level accounting. Sage uniquely allows adaptive queries on overlapping data subsets, only consumes budget of queried blocks, and enables continuous operation on new data.

Streaming DP [9, 17, 19, 20] extends DP to data streams but is restrictive for ML. Data is consumed once and never 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, which is disallowed in streaming DP.

Cummings, et.al. [12] consider *DP over growing databases*. Like us, they run DP algorithms with increasing noise and on exponentially growing data sizes. Unlike us, they require DP algorithms with quantified utility guarantees, which are only available for linear queries and convex ML models. Moreover, their approach has high query runtime, exponential in data dimension!

A few *DP systems* exist, but none focuses on streams or ML. PINQ [35] and its generalization wPINQ [39] give a SQL-like interface to perform DP queries. Airavat [42] gives a MapReduce interface and supports a strong threat model that distrusts the developer. GUPT [36] supports automatic privacy budget allocation and lets programmers specify accuracy goals for arbitrary DP applications. All these works work for static datasets and adopt query-level accounting. This makes them run out of privacy budget even when applied on a continuous data stream.

Local DP (LDP) is another privacy approach in which each data point is collected with DP noise. However LDP is extremely limiting in the computations it can support. It also requires prohibitive amounts of data and specialized algorithms [25]. Combining DP and LDP in a hybrid model can yield better utility [2], but requires specialized algorithms and adaptivity between DP and LDP queries, which is not realistic for general ML.

6 Conclusion

As companies disseminate ML models trained over sensitive data to untrusted domains, it is crucial to start controlling the leakage of the data 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 iterative training with robust validation and its block-level privacy accounting that permits endless operation on streams.

References

[1] 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.

[2] B. Avent, A. Korolova, D. Zeber, T. Hovden, and B. Livshits. BLENDER: Enabling local search with a hybrid differential privacy model. In *Proc. of USENIX Security*, 2017.

[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 ACM Workshop on Artificial Intelligence and Security*, 2015.

[7] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. "You Might Also Like:" Privacy risks of collaborative filtering. In *Proc. of IEEE Symposium on Security and Privacy (S&P)*, 2011.

[8] 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.

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

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

[11] 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.

[12] 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.

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

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

[15] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Minimax optimal procedures for locally private estimation. *Journal of the American Statistical Association*, 113(521):182–201, 2018.

[16] C. Dwork. Differential privacy. In *Automata, languages and programming*, pages 1–12. Springer, 2006.

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

[18] 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.

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2014.

[26] 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.

[27] 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.

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

[29] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[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. Under submission, 2019.

[32] L. E. Li, E. Chen, J. Hermann, P. Zhang, and L. Wang. Scaling machine learning as a service. In *Proc. of The 3rd International Conference on Predictive Applications and APIs*, 2017.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

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

[41] 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.

[42] 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.

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

[44] 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.

[45] A. Smith and A. Thakurta. Differentially private model selection via stability arguments and the robustness of lasso. *Journal of Machine Learning Research (JMLR)*, 30:1–12, 2013.

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

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

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

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.