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 a rich literature in 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 (TFX), the training platform that manages models in production at Google, and illustrate how a systems focus on characteristics of ML workloads can enable 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. 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 their private emails, searches, website visits, heartbeats, and driving behavior. And although ML "code" is derived from sensitive data, we often handle it as we would 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 [6, 27, 37, 48] and sometimes to end-user devices [56, 60] for faster predictions. Some companies also push feature models trained over sensitive data – such as user embedding vectors and statistics of user activity – into model stores. Those stores have been reported to be widely accessible within the companies [27, 37, 53] even though the raw data itself may not be so widely accessible. 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 [8]. Prediction APIs have been shown to be enable testing for membership of a particular user or entry within a training set [54]. 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 incorporate statistics used for featurization – is *fundamentally non-private* [4, 13, 25, 29].

As companies continue to disseminate many versions of models into untrusted domains, it becomes critical to account for and control the data exposure risks posed by these models. We present *Sage*, an ML platform based on TFX that uses differential privacy (DP) [16] to bound the cumulative exposure of individual entries in a company's sensitive data streams through all the ML models the company releases from those streams. At a high level, DP randomizes a computation over a dataset, such as training one ML model, in such a way that the leakage of individual entries in the dataset through the output of the computation is bounded. 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 in DP ML *algorithms* (e.g., [2, 39, 62], 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: (i) control over the quality of the models produced by the DP training pipelines (thereby addressing challenge (2)); and (ii) 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 only recently been explored in DP, and with a purely theoretical and far from practical approach [12]. Most DP literature, largely operating on *individual algorithms*, has focused on 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 ML platforms like TFX.

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*, either 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 that were 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 for training. 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 grows fast enough in terms of adding new blocks relative to the rate at which models are released, Sage will never run out of privacy budget for the stream. Finally, block composition formalizes the most comprehensive adaptivity model in DP literature, supporting both adaptive choice of DP parameters (which has been modeled recently [50] but was typically not supported in DP literature) and adaptive choice of blocks on which to run each training pipeline (which has not been modeled in practical ways before).

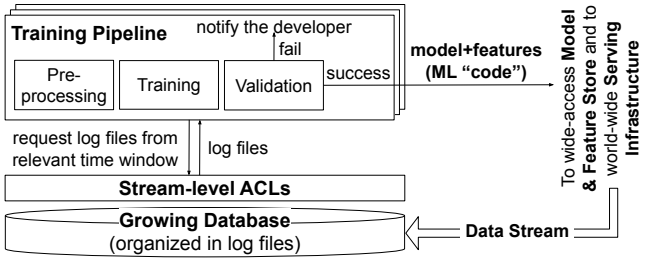


Fig. 1. Typical architecture of an ML platform.

To address the privacy-utility tradeoff, we develop *privacy-adaptive training*, a training procedure that controls the accuracy of DP-trained models by repeatedly and adaptively training them on growing data and/or DP parameters available from the stream. It continues training models until, with high probability (w.h.p.), they meet programmer-specified quality criteria (such as an accuracy or a loss target). privacy-adaptive training makes use of our support for adaptivity in the choice of blocks in the block composition accounting and integrates well with TFX's design, which includes a model evaluation stage in all 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) [6]; Facebook has FBLearner [27]; Uber has Michelangelo [37]; and Twitter has DeepBird [36]. They each operate differently, but they all provide services for model training and serving, plus storage and access control capabilities that constrain raw data accesses. 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 TFX [6], FBLearner [27], and Michelangelo [37]. An 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 *Growing Database* accumulates data from streams, often in log files. Its access control policies are often restrictive for sensitive streams.

The *Training Pipeline* trains a model on data from the Growing Database 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 Growing Database, 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 of those metrics 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. This bundle of model+features is what we consider *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 [53] and tens of thousands of summary statistics [37] 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.

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 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 prediction APIs that can leak training data if queried sufficiently [58]. *Our goal is to "neutralize" the wider exposure of ML codes by making the process of generating them differentially private across all models+features that will ever be 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 the pipeline *as long as the developer is authorized* by the Data Access Control component to access the data stream(s) used by the pipeline. However, we do not trust the wide-access Model and Feature Store and the varied locations to which the serving infrastructure disseminates the model+features. Thus, once a model/feature is pushed to those components, it is considered *released* to untrusted domain and accessible to adversaries.

We focus on two classes of attacks against ML models and statistics (reviewed in Dwork [24]): (1) *tracing attacks* (a.k.a. *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 [4, 25, 29, 54]; 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 [8, 13, 24].

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) [22])**.** 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: for small values, (ϵ, δ)-DP implies that observing one draw from such an algorithm's output gives little information about whether it was run on D or D'. ϵ, often called *privacy budget*, upper bounds the privacy loss from an (ϵ, δ)-DP computation with probability (1-δ). $\oplus$ is a dataset distance metric, such as the symmetric difference [43]; 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 confidentiality attacks in our threat model [8, 24, 30, 54]. 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 a specific output would increase. DP prevents these attacks by ensuring that no specific data point can drastically increase the likelihood of the model being outputted from the training procedure.

DP literature is very rich and mature at this point, including in ML. DP versions exist for almost every popular ML algorithm, including: stochastic gradient descent (SGD) [2, 62]; various regressions [10, 34, 45, 57, 64]; collaborative filtering [42]; language models [41]; feature selection [11]; model selection [55]; evaluation [7]; and statistics, e.g. contingency tables [5], histograms [61]. The privacy module in Tensorflow v2 implements several SGD-based algorithms [39].

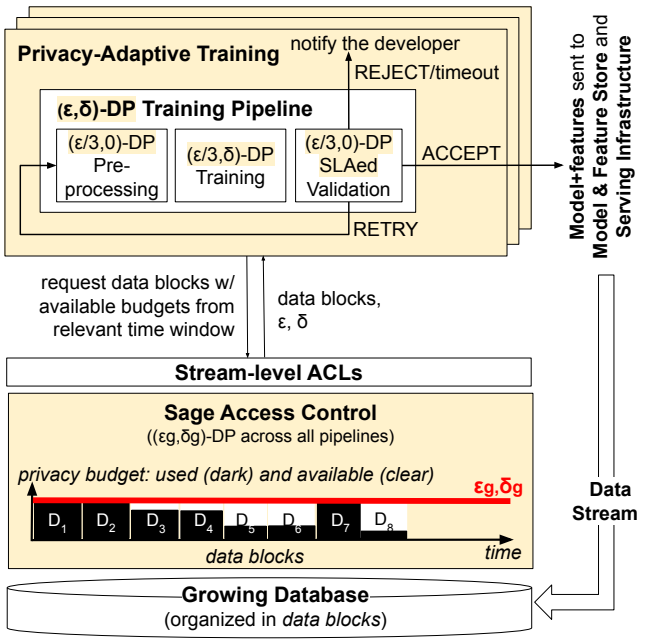


Fig. 2. Sage DP ML Platform. Highlights changes from non-DP version.

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.

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 be applied to user-level privacy (§4), we focus most of our exposition 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 for 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 Sage, 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 *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. The blocks can be defined either 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). It depends on the type of DP semantic one wishes to enforce: event- or user-level privacy, respectively. When the privacy loss for a given block D_i reaches a pre-configured (ϵ_g, δ_g) ceiling, the block is *retired* and will not be used again for training (blocks D_1, D_2, D_7 are retired in Fig. 2). However, new blocks (e.g., new days' worth of data or newly acquired users) from the stream arrive with a clean budget and can be used to train future models. Thus, as long as the database grows fast enough in terms of adding new blocks, the system will never run out of privacy budget for the stream.

Perhaps surprisingly, the preceding design for privacy loss accounting, which we call *block composition* is the first practical approach to avoid running out of privacy budgets while enabling effective training of ML models on growing databases. §3.2 gives the motivation and 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 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.

List. 1 shows the code changes an ML developer would have to make to change a non-DP training pipeline written for

```
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~~sage.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.DNNRegressor~~sage.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.ModelValidator~~sage.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.

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 [46] 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, across the DP function calls. (1) `preprocessing_fn` replaces one call with a DP version that is implemented in Sage:

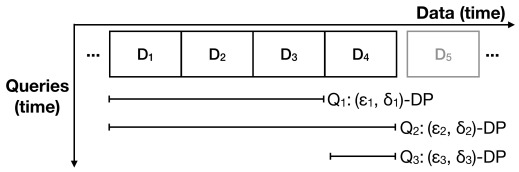


Fig. 3. Characteristics of Data Interaction in ML.

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 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 [43]. 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 formally 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 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, called *block composition*, which leverages characteristics of ML workloads to permit both efficient privacy accounting and efficient learning. §4 formalizes the new theory, which bears broader applications than Sage. This section describes the limitations of existing DP composition for ML workloads and gives the intuition for block composition and how Sage uses it as a new form of access control in ML platforms.

Characteristics of Data Interaction in ML. Fig. 3 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 [43]. There are multiple ways to apply query-level accounting to ML, but each trades off at least one of our requirements.

First, one can permit queries on overlapping data subsets (**R1**) and allow adaptivity across these queries (**R2**) by accounting for composition at query level *against the entire stream*. In Fig. 3, 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 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 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., 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. 3, 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 [41]. 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. Similarly, several DP training algorithms [2, 41] fundamentally rely on sampling small training batches from larger datasets (to amplify privacy), which cannot be done without combining blocks.

Block Composition. Our new composition theory meets all three requirements. It also splits the data stream into non-overlapping *blocks* (e.g., one day's worth of data for event-level privacy), but it allows 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. 3, 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 composition, 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 §3.3) 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 (proof in §4).

The preceding operation resembles a DP-informed retention policy, however one can use block composition to define a broader range of access control policies. Suppose the company is willing to assume that its developers will not collude to violate privacy of its customers – or similarly, that prediction servers or end-user devices situated in distinct geographies will not collude. Then the company could enforce a separate (ϵ_g, δ_g) guarantee for each context (developer or geography).

```
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.

The process will work similarly, except that Sage's Access Control will now maintain separate lists of per-block available budgets for each context, and will deduct privacy budgets only if/when a particular model+features trained on a particular block is made available to that context.

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 give a high-probability guarantee of correctness for the validation (akin to a quality service level agreement, or SLA); and (2) *privacy-adaptive 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. 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 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 which case SLAed validation returns REJECT. We omit the discussion of the reject test here and refer the reader to our extended technical report [35]. 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 metrics, and *absolute errors* of sum-based statistics such as mean and variance. Appendix A details our 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 [35]).

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 [52]. The Loss ACCEPT Test enjoys the following guarantee (proof in Appendix A):

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}*.*

DP Guarantee. ACCEPT and REJECT are $(\epsilon, 0)$-DP (Appendix A). They use disjoint datasets, so `validate` is $(\epsilon, 0)$-DP.

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 [32]. 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 how to do this search. Those that conserve privacy budget have proven the most efficient. Every time a new block is created, its budget is divided evenly between every ML pipeline currently waiting in the system. Allocated DP budget is reserved for the pipeline that received it, but the privacy-adaptive training mechanism 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 between all models still waiting in Sage.

To conserve privacy budget, each pipeline will first train and test using a small configurable privacy 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 compared to alternatives, this iteration strategy conserves privacy budgets and improves performance when multiple Training Pipelines contend for the same blocks.

4 Block Composition Theory

This section serves two purposes: (1) to provide the theoretical backing for block composition, which we invent for Sage but which we believe has broader applications; and (2) to provide an analysis of Sage's use of it to achieve a (ϵ_g, δ_g)-DP access control semantic over data streams.

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 all decisions that can be made in a worst case manner by modeling them as made by an adversary (in simple contexts this worst case guarantee in is contained in the phrase "for any neighboring datasets"). For instance in Fig. 4c, the adversary is allowed to choose the ϵ and δ parameters adaptively, which accounts for user's possibly arbitrary choice of those paramters, but also the fact that past events can influence these choices (i.e. a past change in a data point, the results of previous DP queries, the results of previous DP performance tests...). 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

(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.

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

$\mathcal{A}$ gives k, the index of the block with the adversarially chosen change
for i in $1, \ldots, r$ **do** ▷ ($\mathcal{A}$ depends on $V_1^b, \ldots, V_{i-1}^b$ in iter. i)
  if create new block l and $l == k$ **then**
    $\mathcal{A}$ gives neighboring blocks $\mathcal{D}_k^0$ and $\mathcal{D}_k^1$
  else if create new block l and $l \neq k$ **then**
    $\mathcal{D}_l^b = \mathcal{D}(\mathcal{W}, V_1^b, \ldots, V_{i-1}^b)$
  $\mathcal{A}$ gives blocks_i, (ϵ_i, δ_i), and (ϵ_i, δ_i)-DP Q_i
  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**
    $\mathcal{A}$ receives $V_i^b = Q_i(\bigcup_{j \in \text{blocks}_i} \mathcal{D}_j^b)$
return $V^b = (V_1^b, \ldots, V_r^b)$

(c) Sage Block Composition for Streams and Adaptive Choices.

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 $\leq \epsilon_g$.

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.

(a) 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 $\mathcal{Q}_i$ 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 [33]. 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 [18], strong composition [23], and variations thereof [31]) 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 [18] 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.

(b) Block Composition for Static Datasets: Block composition is designed to improve 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 5), the data is split in blocks, and each DP query runs on a subset of the blocks.

With these changes, 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:*

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

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):

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

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 (Appendix B), 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, thereby violating requirements **R2** and **R3**.

(c) 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 allow training of new ML models, thereby meeting requirement **R3** of endless operation on streams.

Second, interestingly, supporting adaptive choices in the data blocks to query implies supporting adaptive choices in the queries' DP budgets. For a given block, one can express query i's choice whether to use block j as the choice of using (ϵ_i, δ_i) or $(0,0)$ privacy budget. The highlighted portions of lines 8-9 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 [49]. 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 [49]'s Theorem 3.3, we prove:

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, AccessControl$^k_{\epsilon_g,\delta_g}$ enforces:*

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

Proof. Denote l_i the highest block index that existed when query i was run. Denote $D^b_{\leq l_i}$ the data blocks that existed at that time. Recall $v_{<i}$ denotes the results from all queries released previous to i. Query i depends on both $v_{<i}$ and $D^b_{\leq l_i}$; the latter is a random variable that is fully determined by $v_{<i}$. Hence, the privacy loss for Alg. (4c) is: $\text{Loss}(v) = \ln\Big(\prod_{i=1}^{r} \frac{P(V_i^0 = v_i | v_{<i}, D^0_{\leq l_i})}{P(V_i^1 = v_i | v_{<i}, D^1_{\leq l_i})} \Big) = \ln\Big(\prod_{i=1}^{r} \frac{P(V_i^0 = v_i | v_{<i})}{P(V_i^1 = v_i | v_{<i})} \Big)$,

After applying Theorem 4.2, we must show that $\forall k$, $\Big| \ln\Big(\prod_{\substack{i=1 \\ k \in \text{blocks}_i}}^{r} \frac{P(V_i^0 = v_i | v_{<i})}{P(V_i^1 = v_i | v_{<i})} \Big) \Big| \leq \sum_{\substack{i=1 \\ k \in \text{blocks}_i}}^{r} \epsilon_i$ with probability $\geq (1 - \sum_{\substack{i=1 \\ k \in \text{blocks}_i}}^{r} \delta_i)$. The justification follows:

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

Since Q_i is $\epsilon_i(v_{<i}), \delta_i(v_{<i}))$-DP, $\Big| \ln\Big(\frac{P(V_i^0 = v_i | v_{<i})}{P(V_i^1 = v_i | v_{<i})} \Big) \Big| \leq \epsilon_i(v_{<i})$ with probability $\geq 1 - \delta_i(v_{<i})$. Applying a union bound over all queries for which $k \in \text{blocks}_i$ concludes the proof. □

Supporting User-Level Privacy and Other Use Cases. Our block composition theory shows that under the access control scheme described in §3.2 and formalized above, Sage achieves event-level (ϵ_g, δ_g)-DP over each sensitive data stream. However, the theory can be adapted to both user-level privacy and certain use cases on static databases. While in these settings our theory may help do better privacy budget accounting than with previous theory, the system will still run out of privacy budget.

In particular, our theory shows that one can split either 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 loss at the block level. The type of splitting we focus on in Sage is based on time, but the theorems apply to splits based on any attribute whose values are 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. Then block composition gives tighter privacy accounting across these queries than traditional composition.

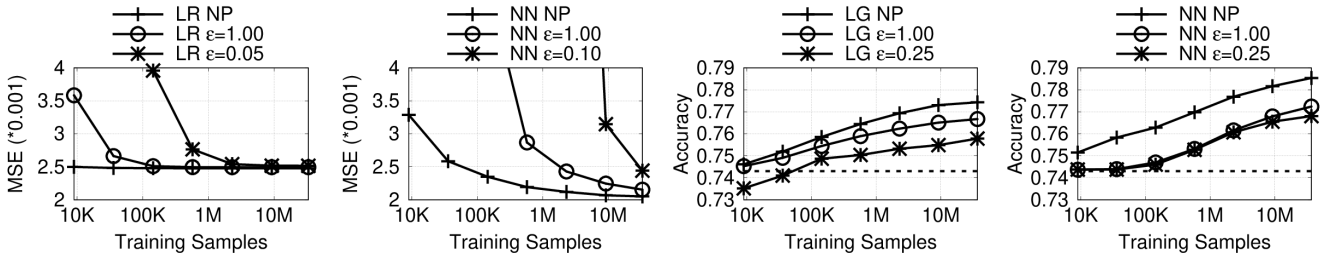
As another example, splitting a stream of user actions by the user ID attribute enables querying or ignoring all observations from a given user, adding support for user-level privacy. However, splitting data over user ID requires some care, since the values of existing user IDs have to be known. If existing user IDs are not known, 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 would enable fine grain DP accounting over queries on any subset 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, and the system needs to add users at a high rate relative to the model release rate to avoid running out of budget. This is unlikely to happen for mature companies, though it may be possible for emerging startups or hospitals, where the stream of incoming users/patients may be high enough to support a modest model workload.

5 Evaluation

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

Methodology. We develop several training pipelines on a 37M-sample dataset from three months of the NYC taxi dataset [46] and a 45M-sample ad impression dataset from Criteo [1]. On the Taxi dataset we consider a regression task to predict the duration of each cab ride using 61 binary features derived from 10 contextual features about each ride. We implement three predictive training pipelines, a *linear regression (LR)*, a *neural network (NN)*, and a *random forest (RF)*, and three summary statistics pipelines (average speeds at three time granularities). On the Criteo dataset we define a binary classification task that predicts if a user will click on an ad using 13 numeric features and 26 categorical features. We implement *logistic regression (LG)* and *neural network (NN)* predictive model pipelines. Tab. 1 details the configurations and parameters of each pipeline.

Training: We make each pipeline DP using known algorithms for the models and the Laplace mechanism for the summary statistics – as indicated in Tab. 1. *Validation:* For the predictive regression models on the taxi dataset, we use the loss SLAed validator with MSE as the loss metric. *Experiments:* Each model is assigned a quality target from a range



(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, 5b, and **??** show the MSE loss on the Taxi regression task (lower is better). 5c, and 5d show the accuracy on the Criteo classification task (higher is better).

Taxi Regression Task		
Models:	**Configuration:**	
Linear Regression (**LR**)	DP Alg.	AdaSSP from [59], (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 $\in [2 \times 10^{-3}, 7 \times 10^{-3}]$
Statistics:	**Configuration:**	
Avg.Speed x3*	Targets	Absolute error $\in \{1, 5, 7.5, 10, 15\}$ km/h
Criteo Classification Task		
Models:	**Configuration:**	
Logistic Regression (**LG**)	DP Alg.	DP SGD from [40], (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 [40], (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 $\in [0.74, 0.78]$

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

of possible values. For the regression task, a naïve model that always returns the average ride duration has an MSE of 0.0069, so we configure quality targets strictly smaller than this value. For the summary statistics we use our sum-based SLAed validator, with absolute error as the metric. On the the classification task, a naïve model that always predicts the more common non-click label will have an accuracy of 74.3%. We use the value as the lower bound of the accuracy targets. 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, 41]. Where DP budgets must be fixed, we use values indicated in Tab. 1 which correspond to a large budget ($\epsilon = 1$), and a small but achievable budget that varies across tasks and models. Other defaults: 90%::10% train::test ratio; $\eta = 0.05$; $\delta = 10^{-6}$.

5.1 Unreliability of DP Training Pipelines in TFX (Q1)

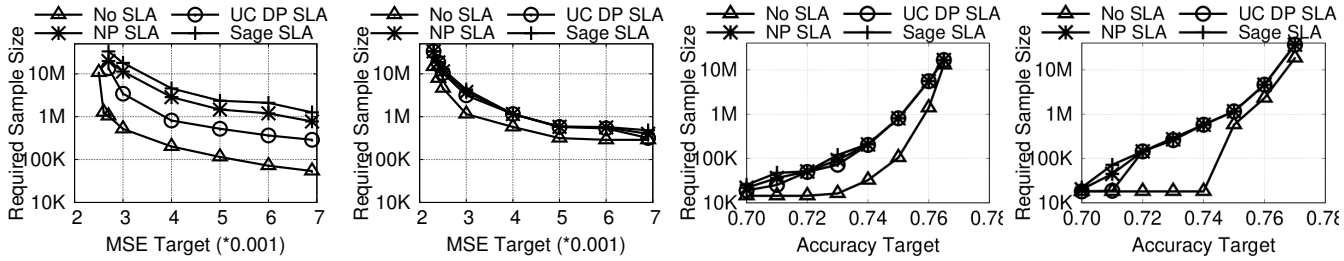
We first evaluate DP's impact on model training. Fig. 5 shows loss or accuracy of each model when trained on increasing amounts of training data and evaluated on 100K held out samples from their respective datasets. Three versions are shown for each model: the non-DP version (NP), a large DP budget version ($\epsilon = 1$), 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 outperform the linear models 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 impact on validation, we train and validate our models for both tasks, with and without DP. We use TFX's vanilla validators, which simply 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 heldout 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 in the amount of data needed compared to a non-DP test. We evaluate reliability and sample complexity for the ACCEPT test of SLAed validation.

Tab. 2 shows the fraction of ACCEPTed models that violate their quality targets when re-evaluated on the 100K-sample held-out set. For three 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 hope to achieve with statistical confidence. (3) *UC DP SLA*, a DP SLAed validation but without the correction for DP impact. (4) *Sage SLA*, our DP SLAed validator, with



(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, 6b, and **??**) 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.

Taxi Loss SLA				
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

Criteo Accuracy SLA				
η	**No SLA**	**NP SLA**	**UC DP SLA**	**Sage SLA**
0.01	0.2515	0.0052	0.0544	0.0018
0.05	0.2515	0.0065	0.0556	0.0023
0.10	0.2515	0.0065	0.0560	0.0029

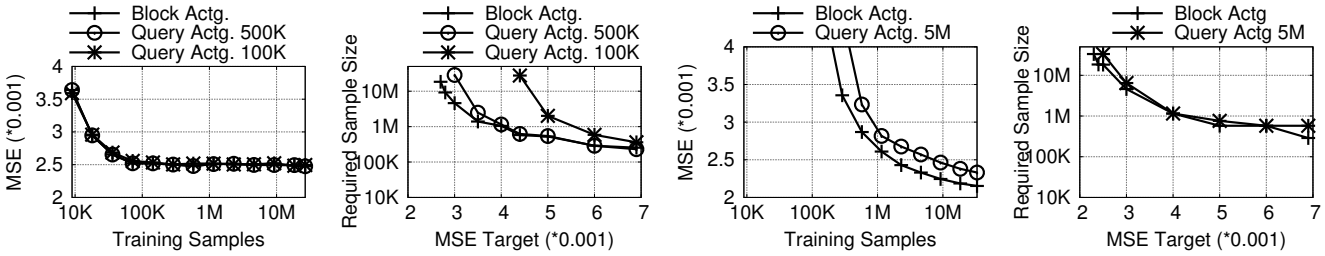
Tab. 2. Target Violation Rate of ACCEPTed Models. Violations are across all models separately trained with privacy-adaptive training.

correction. We make three observations. First, the NP SLA violation rates are much lower than the configured η values for both the accuracy and the loss SLA, because the statistical tests we use are conservative. Second, Sage's DP-corrected validation accepts models with violation rates that are close to the NP SLA. Slightly higher in the case of the loss SLA and slightly lower in the case of the accuracy SLA, but *well below the configured error rate for both*. 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 (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 to a non-DP test. This new data is supplemented by Sage's privacy-adaptive training. Fig. 6a, 6b, and **??** 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, UC DP SLA) requires the least data when it has a high failure rate. This is because it erroneously accepts models on small sample sizes. Second, Sage's SLA validation accepts models with targets that are very close to the values that are accepted (potentially erroneously) 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 to the validation but no privacy (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 more or fewer 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 DNN models on the Taxi dataset, when operating on blocks of different sizes, 100K and 500K for the LR, and 5M for the DNN. 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 or 500K, and 0.0044 for blocks of size 500K. 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 DNN 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 one to an MSE target of 0.0025 (against Sage's 0.0023), and requires twice as much data as with block composition.



(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.

5.4 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 from the Taxi dataset arriving over discrete time steps. At each step, 16K new data arrives and forms a new block corresponding to approximately 1 hour of data from the Taxi dataset. The job arrival rate is generated by a Gamma process. A new model's size if drawn from a power law distribution in job size (number of data points required to reach the target). Given an arriving pipeline's size, if is drawn from the relevant subset of our 25 configurations of Taxi tasks and targets (see Tab. 1).

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 four privacy-adaptive training strategies of DP resource allocation. 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. Second, *Query Accounting* allows concurrent training but requires that blocks be queried independently resulting in significantly higher sample complexities as shown in § 5.3. And finally, two privacy-adaptive training strategies take advantage of Sage's block composition, 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 its budget. *Concurrent/Conserve (Sage)* uses the strategy defined in § 3.3. *Concurrent/Aggressive* uses as much privacy budget as is available when a pipeline is invoked.

Fig. 8 shows how the average model release time in steady state for each strategy, under increasing load and as the system enforces $(\epsilon_g, \delta_g) = (1.0, 10^{-6})$-DP over the entire stream.

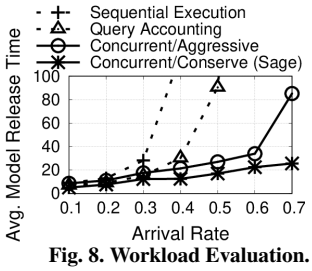


Fig. 8. Workload Evaluation.

We make two observations. First, Sage's block composition and the concurrent training it enables, is crucial: the Sequential Execution and Query Accounting quickly degrade to off-the-charts release times. 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 withing 24hours. 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 and "Concurrent/Conserve" has a release time 4x smaller than "Concurrent/Aggressive." Privacy budget conservation reduces the amount of privacy budget that will be consumed by an individual pipeline, thus allowing new pipelines to use the remaining budget when they arrive in the system.

6 Related Work

Sage's main contribution – block composition – is related to *DP composition theory*. Basic [18] and strong [23, 31] composition theorems give the DP guarantee for multiple queries with adaptively chosen computation. McSherry [43] and Zhang, et.al. [63] show that non-adaptive queries over non-overlapping subsets of data can share the DP budget. Rogers, et.al. [50] analyze composition under adaptive DP parameters, which is crucial to our block-level accounting. These works all consider fixed datasets and query-level accounting.

Compared to all these works, our major contribution is to formalize the new block-level DP interaction model, which enables training ML models 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 which works only on current data. ML workloads require different interaction with data: new data is incorporated into training, and historical data is revisited to allow training on large datasets. In proving our new interaction model DP, we leverage prior theoretical DP results and analysis methods. However, among the four key properties of our interaction model (Fig. 4c), namely that (1) queries interacting with subsets of blocks; (2) that the blocks' data is impacted by previous queries; (3) that queried blocks are

adaptively chosen; and (4) that DP parameters are adaptively chosen, only property (4) was explicitly addressed in [50].

Streaming DP [9, 17, 19, 20] 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. [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; those are available only for linear queries and convex ML models. Moreover, their query runtimes are exponential in data dimension!

A few *DP systems* exist, but none focuses on streams or ML. PINQ [43] and its generalization wPINQ [47] give a SQL-like interface to perform DP queries. They introduce the partition operator that allows *parallel composition*, which resembles Sage's block composition. However, the partition operator only supports non-adaptive parallel computations on non-overlapping partitions, which is insufficient for ML. Airavat [51] gives a MapReduce interface and supports a strong threat model that distrusts the developer. Through their integration with mandatory access control systems, they adopt a similar perspective of DP as a form of access control as us. GUPT [44] 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 were unused data were available. Block-level accounting avoids this severe limitation but applies to workloads with specific data interaction characteristics (§3.2).

Local DP (LDP) is another privacy approach in which each data point is collected with DP noise. It gives compelling privacy guarantees but supports very limited computations, such as histograms and statistical queries. It also requires prohibitive amounts of data [26]. Combining DP and LDP can yield better utility [3] but requires specialized algorithms and adaptivity between DP and LDP queries.

Finally, we justify our decision to use DP to control leakage of sensitive information through ML models. Other approaches include: (1) sensitive data scrubbing, as proposed to prevent memoization of secrets in language models [8]; and (2) using homomorphic encryption to perform inference on encrypted models at untrusted locations. Neither is satisfactory for our threat model (§2.2). The former (1) does not guarantee protection especially when many models and statistics are pushed to production. The latter (2) does not protect against black-box confidentiality attacks: models have been shown to leak information just through their predictions, which ultimately need to be decrypted to be shown to users.

7 Limitations and Future Work

Event-Level Privacy. Motivated by our desire to overcome difficult challenges of global DP – including running out of privacy budget, which we believe is a real roadblock to DP adoption – we focused this paper and evaluation on event-level DP, for which we believe these challenges can indeed be addressed practically. However, we acknowledge that this semantic has significant limitations when groups of correlated observations can reveal sensitive information. The best known example of such correlated observations happens when users of an application each contribute multiple observations. In this case all observations corresponding to the same user are correlated, and can reveal sensitive information such as demographic attributes of the user. Other examples include repeated measurements of a phenomenon over time, or users and their friends being correlated on social networks.

Even in the face of correlated data, the DP holds for each individual observation: other correlated observations constitute side information, to which DP is known to be resilient. Of course, event-level DP will not prevent an attacker from learning about groups of data points (e.g. user features). Event-level DP provides some protection to groups of observations through group privacy [21], but these guarantees degrade exponentially with group size. An attacker could thus learn information about large groups of correlated observations.

Using semantics such as user-level DP, which enforces the DP guarantee from Definition 2.1 over all observations of a user, would provide stronger guarantees of privacy and data protection. §4 includes a discussion of how user-level privacy can be supported with our theory and system design. The challenge is that the central requirement for not running out of privacy budget in DP – that the database grows fast enough in terms of new blocks – is unlikely to be met with a user-level semantic, especially for mature companies.

DP Budget Management. A key practical question when training DP models is to choose the DP parameters (ϵ, δ) and number of data points to allocate to each model, to conserve privacy budget while reaching quality targets. Sage proposes and evaluates a heuristic to perform this allocation (§3.3, §5.4) but identifying principled approaches to perform these allocations remains an open problem. We suspect that resource allocation literature from systems will be relevant.

Furthermore, Sage impacts traditional resources. Some DP training techniques can be slower than their non DP equivalent. For instance, DP SGD involves sampling batches without replacement which is slower than the typical shuffle and loop approach used in deep learning. Computing gradients for DP updates also requires more information, which hurts performance. More specific to Sage's approach, Privacy-Adaptive Training requires training each model multiple times. The management and allocation of compute resources in the context of Sage is a promising direction for systems support of DP ML workloads.

8 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 block-level accounting that permits endless operation on streams and its privacy-adaptive training that adds reliability to DP training pipelines.

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] 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.

[4] 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.

[5] 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.

[6] 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.

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

[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 and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 1963.

[29] 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.

[30] B. Jayaraman and D. Evans. Evaluating differentially private machine learning in practice. In *USENIX Security Symposium (USENIX Security 19)*, 2019.

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

[32] 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.

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

[34] 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.

[35] M. Lecuyer, R. Spahn, K. Vodrahalli, R. Geambasu, and D. Hsu. Privacy accounting and quality control in the sage differentially private ml platform. Technical report, 2019.

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

[37] 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.

[38] A. Maurer and M. Pontil. Empirical Bernstein Bounds and Sample Variance Penalization. 2009.

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

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

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

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

[47] 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.

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

[49] 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.

[50] 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.

[51] 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.

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

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

[54] 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.

[55] 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.

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

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

[58] 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.

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

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

A Validation Tests

Sage has built-in validators for three classes of metrics. §3.3 describes the high level approach and the specific functionality and properties of the loss-based validator. This section details all three validators and proves their statistical and DP guarantees.

A.1 SLAed Validator for Loss Metrics

Denote a loss function $l(f, x, y)$ with range $[0, B]$ measuring the quality of a prediction $f(x)$ with label y (lower is better), and a target loss τ_{loss} on the data distribution $\mathcal{D}$.

ACCEPT Test: Given a the DP-trained model f^{dp}, we want to release f^{dp} only if $\mathcal{L}_{\mathcal{D}}(f^{\text{dp}}) \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}} l(f^{\text{dp}}, x, y) \leq \tau_{loss}$. The test works as follows. First, 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 11-13 List. 2):

$$\underline{n}_{\text{te}}^{\text{dp}} = n_{\text{te}} + \text{Laplace}(\frac{2}{\epsilon}) - \frac{2}{\epsilon}\ln(\frac{3}{2\eta}).$$

Then, compute a DP estimate of the loss corrected for DP impact (Lines 14-17 List. 2) to be an upper bound on the true value, $\mathcal{L}_{te}(f^{\text{dp}}) \triangleq \frac{1}{n_{\text{te}}}\sum_{te} l(f^{\text{dp}}, x, y)$, with probability $(1 - \frac{\eta}{3})$:

$$\overline{\mathcal{L}}_{te}^{dp}(f^{\text{dp}}) = \frac{1}{\underline{n}_{\text{te}}^{\text{dp}}}\Big(\sum_{te} l(f^{\text{dp}}, x, y) + \text{Laplace}(\frac{2B}{\epsilon}) + \frac{2B}{\epsilon}\ln(\frac{3}{2\eta})\Big)$$

Lines 18-20, we see that Sage will ACCEPT when:

$$\overline{\mathcal{L}}_{te}^{dp}(f^{\text{dp}}) + \sqrt{\frac{2B\overline{\mathcal{L}}_{te}^{dp}(f^{\text{dp}})\ln(3/\eta)}{\underline{n}_{\text{te}}^{\text{dp}}}} + \frac{4B\ln(3/\eta)}{\underline{n}_{\text{te}}^{\text{dp}}} \leq \tau_{loss}.$$

This test gives the following guarantee:

Proposition A.1 (Loss ACCEPT Test (same as Proposition 3.1))**.** *With probability at least* $(1 - \eta)$*, the* Accept *test returns true only if* $\mathcal{L}_{\mathcal{D}}(f^{\text{dp}}) \leq \tau_{loss}$.

Proof. The corrections for DP noise imply that $P(\underline{n}_{\text{te}}^{\text{dp}} > n_{\text{te}}) \leq \frac{\eta}{3}$, and $P(\mathcal{L}_{te}^{dp}(f^{\text{dp}}) > \mathcal{L}_{te}(f^{\text{dp}})) \leq \frac{\eta}{3}$ (i.e. the lower bounds hold with probability at least (1-$\frac{\eta}{3}$)). Define $\text{UB}^{dp} \triangleq \overline{\mathcal{L}}_{te}^{dp}(f^{\text{dp}}) + \sqrt{\frac{2B\overline{\mathcal{L}}_{te}^{dp}(f^{\text{dp}})\ln(3/\eta)}{\underline{n}_{\text{te}}^{\text{dp}}}} + \frac{4\ln(3/\eta)}{\underline{n}_{\text{te}}^{\text{dp}}}$, and $\text{UB} \triangleq \mathcal{L}_{te}(f^{\text{dp}}) + \sqrt{\frac{2B\mathcal{L}_{te}(f^{\text{dp}})\ln(3/\eta)}{n_{\text{te}}}} + \frac{4\ln(3/\eta)}{n_{\text{te}}}$. Applying Bernstein's inequality [52] yields $P(\mathcal{L}_{\mathcal{D}}(f^{\text{dp}}) > \text{UB}) \leq \frac{\eta}{3}$. A union bound on those three inequalities gives that with probability at least $(1-\eta)$, $\mathcal{L}_{\mathcal{D}}(f^{\text{dp}}) \leq \text{UB} \leq \text{UB}^{dp}$. The test ACCEPTs when $\text{UB}^{dp} \leq \tau_{loss}$. □

This test uses Bernstein's concentration inequality to compute a upper bound for the loss over the entire distribution [52], which will give good bounds if the loss is small. If instead one expects the variance to be small, one can use empirical Bernstein bounds [38] as a drop-in replacement. Otherwise, one can fall back to Hoeffding's inequality [28].

REJECT Test: The REJECT test terminates a model when no model of the considered class $\mathcal{F}$ can hope to achieve the desired τ_{loss} performance. Noting the best possible model on the data distribution $f^\star \triangleq \arg\min_{f\in\mathcal{F}} \mathcal{L}_{\mathcal{D}}(f)$, we want to reject when $\mathcal{L}_{\mathcal{D}}(f^\star) > \tau_{loss}$. To do this, it considers the best model in $\mathcal{F}$ on the training set $\hat{f} \triangleq \arg\min_{f\in\mathcal{F}} \mathcal{L}_{tr}(f)$, and proceeds as follows. First, compute the DP upper and lower bounds for n_{tr}, holding together with probability at least $(1-\frac{\eta}{3})$ (Lines 24-27 List. 2):

$$n_{\text{tr}}^{\text{dp}} = n_{\text{tr}} + \text{Laplace}(\frac{2}{\epsilon}),$$
$$\underline{n}_{\text{tr}}^{\text{dp}} = n_{\text{tr}}^{\text{dp}} - \frac{2}{\epsilon}\ln(\frac{3}{\eta}),$$
$$\overline{n}_{\text{tr}}^{\text{dp}} = n_{\text{tr}}^{\text{dp}} + \frac{2}{\epsilon}\ln(\frac{3}{\eta}).$$

Then, compute a DP estimate of the loss corrected for DP impact (Lines 14-17 List. 2) to be a lower bound on the true value with probability $(1-\frac{\eta}{3})$:

$$\underline{\mathcal{L}}_{te}^{dp}(\hat{f}) = \frac{1}{\overline{n}_{\text{tr}}^{\text{dp}}}\Big(\sum_{tr} l(\hat{f},x,y) + \text{Laplace}(\frac{2B}{\epsilon}) - \frac{2B}{\epsilon}\ln(\frac{3}{2\eta})\Big).$$

Finally, Sage will REJECT when:

$$\underline{\mathcal{L}}_{te}^{dp}(\hat{f}) - B\sqrt{\frac{\log(3/\eta)}{\underline{n}_{\text{tr}}^{\text{dp}}}} > \tau_{loss}.$$

This test gives the following guarantee:

Proposition A.2 (Loss REJECT Test (same as Proposition **??**))**.** *With probability at least* $(1-\eta)$*,* f^{dp} *(or more accurately* $\mathcal{F}$*) is rejected only if* $\mathcal{L}_{\mathcal{D}}(f^\star) > \tau_{loss}$*.*

Proof. By definition, $\mathcal{L}_{tr}(\hat{f}) \leq \mathcal{L}_{tr}(f^\star)$. Applying Hoeffding's inequality [28] to $f^\star$ gives $P(\mathcal{L}_{\mathcal{D}}(f^\star) < \mathcal{L}_{tr}(f^\star) - B\sqrt{\frac{\log(3/\eta)}{n_{\text{tr}}}}) \leq \frac{\eta}{3}$. Similarly to the proof for Proposition (A.1), applying a union bounds over this inequality and the DP correction gives that with probability at least $(1-\eta)$, $\mathcal{L}_{\mathcal{D}}(f^\star) \geq \mathcal{L}_{tr}(f^\star) - B\sqrt{\frac{\log(3/\eta)}{n_{\text{tr}}}} \geq \mathcal{L}_{tr}^{dp}(\hat{f}) - B\sqrt{\frac{\log(3/\eta)}{\underline{n}_{\text{tr}}^{\text{dp}}}} > \tau_{loss}$, concluding the proof. □

Here we leverage Hoeffding's inequality [28] as we need to bound $\mathcal{L}_{\mathcal{D}}(f^\star)$: since we do not know $f^\star$, we cannot compute its variance or loss on the training set to use (empirical) Bernstein bounds.

We highlight that this result relies on $\hat{f}$, which may not always be available. In particular, is can be computed for convex problems, but not for DNNs for instance. In practice, we use f^{dp} as an approximation, which is a heuristic as we do not know how to correct for the worst case DP noise impact on training in general, and can lead to false rejections (§5.2).

DP Guarantee: We now need to prove that the ACCEPT and REJECT tests each are $(\epsilon, 0)$-DP. Since they each use a disjoint split of the dataset (training and testing) running both tests is $(\epsilon, 0)$-DP over the entire dataset.

Proposition A.3 (DP ACCEPT)**.** Accept *is* $(\epsilon, 0)$*-DP.*

Proof. The only data interaction for ACCEPT are to compute $\underline{n}_{\text{te}}^{\text{dp}}$ and $\overline{\mathcal{L}}_{te}^{dp}(f^{\text{dp}})$. The former is sensitivity 1, and is made $\frac{\epsilon}{2}$-DP with the Laplace mechanism. The latter uses data in the inner summer, which is sensitivity B and is made $\frac{\epsilon}{2}$-DP with the Laplace mechanism. Using basic composition, the test is $(\epsilon, 0)$-DP. □

The proof for REJECT is a bit more complicated as part of the procedure involves computing $\hat{f}$, which is not DP.

Proposition A.4 (DP REJECT)**.** Reject *is* $(\epsilon, 0)$*-DP.*

Proof. To apply the same argument as for Prop.A.3, we need to show that computing $\mathcal{L}_{tr}(\hat{f})$, which includes computing $\hat{f}$, has sensitivity B. Recall that $\hat{f}$ minimizes the training loss: $\hat{f}_{tr} = \arg\min_{f\in\mathcal{F}} \sum_{n_{\text{tr}}} l(f,x,y)$. If we add a data point d, then $\hat{f}_{tr}$ has a loss at worst B on the new point. Because the best model with the new data point $\hat{f}_{tr+d}$ is a least as good as $\hat{f}_{tr}$ (otherwise it wouldn't be the arg min model), $\mathcal{L}_{tr+d}(\hat{f}_{tr+d}) \leq \mathcal{L}_{tr}(\hat{f}_{tr}) + B$. Similarly, $\hat{f}_{tr+d}$ cannot be better than $\hat{f}_{tr}$ on the training set, so $\mathcal{L}_{tr+d}(\hat{f}_{tr+d}) \geq \mathcal{L}_{tr}(\hat{f}_{tr})$. Hence the sensitivty of computing $\mathcal{L}_{tr}(\hat{f})$ is at most B. □

Finding $\hat{f}$ is possible for some classes of problems (e.g. convex optimization), but not all (e.g. DNNs). As we see in the proof of Prop.A.4, getting the loss minimizer $\hat{f}$ is crucial to the DP property. Using f^{dp} as an approximation, as we do is Sage, also ensures that the procedure remains DP when finding $\hat{f}$ is not possible.

A.2 SLAed Validator for Accuracy

The accuracy metric applies to classification problems, where a model predicts one out of many classes. The target τ_{acc} is the proportion of correct predictions to reach on the

data distribution. The accuracy validator follows the same logic as the loss validator, with two small changes. First, the upper and lower bounds are reversed as losses are minimized while accuracy is maximized. Second, the result of classification predictions follows a binomial distribution, which yields tighter confidence intervals. Note $\mathbb{1}\{\text{predicate}\}$ the indicator function with value 1 if the predicate is true, and 0 otherwise, and $\overline{\text{Bin}}(k, n, \eta)$ and $\underline{\text{Bin}}(k, n, \eta)$ the upper and lower bounds on the probability parameter p of a binomial distribution such that k happen can happen with probability at least η out of n independent draws (both can be conservatively approximated using a Clopper-Pearson interval).

We compute $k_{te}^{dp} = \sum_{n_{\text{te}}} \mathbb{1}\{f(x) = y\} + Laplace(\frac{2}{\epsilon})$ and $n_{\text{te}}^{\text{dp}} = n_{\text{te}} + Laplace(\frac{2}{\epsilon})$. Similarly for the training set tr, $k_{tr}^{dp} = \sum_{n_{\text{te}}} \mathbb{1}\{\hat{f}(x) = y\} + Laplace(\frac{2}{\epsilon})$ and $n_{\text{tr}}^{\text{dp}} = n_{\text{tr}} + Laplace(\frac{2}{\epsilon})$. Sage will ACCEPT when:

$$\underline{\text{Bin}}\left(k_{te}^{dp} - \frac{2}{\epsilon}\ln(\frac{3}{\eta}), n_{\text{te}}^{\text{dp}} + \frac{2}{\epsilon}\ln(\frac{3}{\eta}), \frac{\eta}{3}\right) \geq \tau_{acc}.$$

And REJECT when:

$$\overline{\text{Bin}}\left(k_{tr}^{dp} + \frac{2}{\epsilon}\ln(\frac{3}{\eta}), n_{\text{tr}}^{\text{dp}} - \frac{2}{\epsilon}\ln(\frac{3}{\eta}), \frac{\eta}{3}\right) < \tau_{acc}.$$

Using the same reasoning as before, we can prove the equivalent four Propositions (A.1, A.2, A.3, A.4) for the accuracy validator. Finding $\hat{f}$ for accuracy is computationally hard.

A.3 SLAed Validator for Sum-based Statistics

This validator applies to computing sum based statistics (e.g. mean, variance). The target is defined as the maximum size of the absolute (additive) error τ_{err} for these summary statistics on the data distribution. This error can be computed directly on the training set, so there is no need to a test set. Second, because of the law of large numbers, we can always reach the target precision, so there is no rejection test. We next show the test using Hoeffding's inequality [28] (if we expect the variance to be small, empirical Bernstein bounds [38] will be better and are directly applicable).

Compute:

$$n_{\text{tr}}^{\text{dp}} = n_{\text{tr}} + Laplace(\frac{2}{\epsilon}) - \frac{2}{\epsilon}\ln(\frac{2}{\eta})$$

SageACCEPTs if:

$$\frac{1}{n_{\text{tr}}^{\text{dp}}}\frac{2}{\epsilon}\ln(\frac{2}{\eta}) + B\sqrt{\frac{\ln(2/\eta)}{n_{\text{tr}}^{\text{dp}}}} \leq \tau_{err},$$

in which case the absolute error is bellow τ_{err} with probability at least $(1 - \eta)$, accounting for the statistical error, as well as the impact of DP noise on both the sum based summary statistic and the ACCEPT test. Once again, the same reasoning allows us to prove equivalent Propositions to A.1 and A.3.

B Block Composition

This section makes several clarifications and precisions to the block composition theory in §4. It then ports prior strong composition results to our block accounting model, both for fixed and adaptive choices of blocks and DP parameters.

B.1 Clarifications and Precisions

Neighboring Datasets. We measure the distance between datasets using the symmetric difference: viewing a dataset as a multiset, twp datasets D and D' are neighboring if their disjunctive union, the elements which are in one of the sets but not in their intersection, is at most one. This definition is not the most standard: most DP work uses the Hamming distance which counts the number of records to change to go from D to D'. Intuitively, under the symmetric difference an attacker can add or remove a record in the dataset. Changing a record corresponds to a symmetric difference of size 2, but a Hamming distance of 1. Changing a record can still be supported under the symmetric difference using group privacy [22]: it is thus slightly weaker but as general.

We chose to use the symmetric difference following PINQ [43], as this definition is better suited to the analysis of DP composition over splits of the dataset (our blocks). Changing one record can indeed affect two blocks (e.g. the timestamp is changed) while adding or removing records can only affect one block.

Neighboring Streams. This notion of neighboring dataset extends pretty directly to streams of data [9, 17, 19, 20]. Two streams D and D' indexed by t are neighboring if there exists an index T such that: for $t < T$ the streams are identical (i.e. $|D_{t<T} \oplus D'_{t<T}| = 0$), and for all $t \geq T$ the streams up to t form neighboring datasets (i.e. $|D_{t\geq T} \oplus D'_{t\geq T}| \leq 1$). This is equivalent to our Algorithm (4b) where the data, though unknown, is fixed in advance with only one record being changed between the two streams.

This definition however is restrictive, because a change in a stream's data will typically affect future data. This is especially true in an ML context, where a record changed in the stream will change the targeting or recommendation algorithms that are trained on this data, which in turn will impact the data collected in the future. Because of this, D and D' will probably differ in a large number of observations following the adversary's change of one record. Interactions described in our Algorithm (4c) model these dependencies. We show in Theorem (4.3) that if the data change impacts future data only through DP results (e.g. the targeting and recommendation models are DP) and mechanisms outside of the adversary's control (our world variable W), composition results are not affected.

Privacy Loss Semantics. Recall that bounding the privacy loss (Definition 4.1) with high probability implies DP [33]: if with probability $(1 - \delta)$ over draws from $v \sim V^0$ (or $v \sim V^1$) $|\text{Loss}(v)| \leq \epsilon$, then the interaction generating V^b is (ϵ, δ)-DP.

In this paper, we implicitly treated DP and bounded loss as equivalent by declaring Q_is as (ϵ, δ)-DP, but proving composition results using a bound on Q_i's privacy loss. However, this is not exactly true in general, as (ϵ, δ)-DP implies a bound on

privacy loss with weaker parameters, namely that with probability at least $(1 - \frac{2\delta}{\epsilon e^{\epsilon}})$ the loss is bounded by 2ϵ. In practice, this small difference in not crucial, as the typical Laplace and Gaussian DP mechanisms (and those we use in Sage) do have the same (ϵ, δ)-DP parameters for their bounds on privacy loss [21]: the Laplace mechanism for $(\epsilon, 0)$-DP implies that the privacy loss is bounded by ϵ and achieving (ϵ, δ)-DP with the Gaussian mechanism implies that the privacy loss is bounded by ϵ with probability at least $(1 - \delta)$.

B.2 Strong Composition with Block-Level Accounting

We now prove strong composition results for Algorithms (4b) and (4c).

Fixed Blocks and DP Parameters: We now show how to use advanced composition results (e.g. [23]) is the context of block composition. This approach requires that both the blocks used by each query and the DP parameters are fixed in advanced, and correspond to Algorithms (4b).

Theorem B.1 (Strong Block Composition (fixed DP parameters))**.** *BlockCompose($\mathcal{A}$, b, r, $(\epsilon_i, \delta_i)_{i=1}^r$, $blocks_{i=1}^r$) is (ϵ_g, δ_g)-DP, with:*

$$\epsilon_g = \max_k \Big(\sum_{\substack{i=1 \\ k \in blocks_i}}^{r} (e^{\epsilon_i} - 1)\epsilon_i + \sqrt{\sum_{\substack{i=1 \\ k \in blocks_i}}^{r} 2\epsilon_i^2 \log(\frac{1}{\tilde{\delta}})} \Big),$$

$$\delta_g = \tilde{\delta} + \max_k \Big(\sum_{\substack{i=1 \\ k \in blocks_i}}^{r} \delta_i \Big)$$

Proof. After applying Theorem 4.2, what remains to be shown is that $\forall k$, $\Big|\ln\Big(\prod_{\substack{i=1 \\ i \in \text{blocks}_k}}^{r} \frac{P(V_i^0 = v_i | v_{<i})}{P(V_i^1 = v_i | v_{<i})}\Big)\Big| \leq \sum_{\substack{i=1 \\ k \in blocks_i}}^{r} (e^{\epsilon_i} - 1)\epsilon_i + \sqrt{\sum_{\substack{i=1 \\ k \in blocks_i}}^{r} 2\epsilon_i^2 \log(\frac{1}{\tilde{\delta}})}$, with probability at least $(1 - \tilde{\delta} \sum_{\substack{i=1 \\ k \in \text{blocks}_i}}^{r} \delta_i)$. Using the fact Q_i's privacy loss is bounded by ϵ_i with probability at least $(1 - \delta_i)$, we know that there exists events E_i and E_i' with joint probability at least $(1 - \delta_i)$ such that for all v_i, $\Big|\ln\Big(\frac{P(V_i^0 = v_i | E_i)}{P(V_i^1 = v_i | E_i')}\Big)\Big| \leq e^{\epsilon}$. We can now condition the analysis on E_i and E_i', and use Theorem 3.20 of [21] to get that with probability at least $(1 - \tilde{\delta})$, $\Big|\ln\Big(\frac{P(V^0 = v | E_i)}{P(V^1 = v | E_i')}\Big)\Big| \leq e^{\epsilon_k}$, where $\epsilon_k = \sum_{\substack{i=1 \\ k \in blocks_i}}^{r} (e^{\epsilon_i} - 1)\epsilon_i + \sqrt{\sum_{\substack{i=1 \\ k \in blocks_i}}^{r} 2\epsilon_i^2 \log(\frac{1}{\tilde{\delta}})}$. A union bound on the E_i and E_i' for all $\{i, k \in blocks_i\}$ completes the proof. □

The proof directly extends to the stream setting (yellow parts of Alg. (4b) in the same way as in the proof of Theorem 4.3.

Adaptive Blocks and DP Parameters: Recall that with either adaptive blocks or DP parameters (or both), Note the fact that DP parameters $(\epsilon_i(v_{<i}, \delta_i(v_{<i})$ depend on history, which is why traditional composition theorems do not apply to the adaptive parameter case. For basic composition, the DP parameters still "sum" under composition. However, as showed in [49], strong composition yields a different formula: while the privacy loss still scales as the square root of the number of queries, the constant is worse than with parameters fixed in advance.

Theorem B.2 (Strong Adaptive Stream Block Composition)**.** *AdaptiveStreamBlockCompose($\mathcal{A}$, b, r, ϵ_g, δ_g, $\mathcal{W}$) is (ϵ_g, δ_g)-DP, and:*

$$\max_k \Big(\sum_{\substack{i=1 \\ k \in \text{blocks}_i}}^{r} \frac{(e^{\epsilon_i} - 1)\epsilon_i}{2} + \sqrt{2\Big(\sum_{\substack{i=1 \\ k \in \text{blocks}_i}}^{r} \epsilon_i^2 + \frac{\epsilon_g^2}{28.04 \log(1/\tilde{\delta})} \Big)} \sqrt{\Big(1 + \frac{1}{2}\log\Big(\frac{28.04 \log(1/\tilde{\delta}) \sum_{\substack{i=1 \\ k \in \text{blocks}_i}}^{r} \epsilon_i^2}{\epsilon_g^2} + 1\Big)\log(\frac{1}{\tilde{\delta}})\Big)} \Big) \leq \epsilon_g,$$

$$\tilde{\delta} + \max_k \Big(\sum_{\substack{i=1 \\ k \in \text{blocks}_i}}^{r} \delta_i \Big) \leq \delta_g$$

Proof. Similarly to the proof of Theorem 4.3, we apply Theorem 4.2, and bound the privacy loss of any block k using Theorem 5.1 of [49]. □