Private Transferrable Knowledge: A New Data Protection Abstraction for Machine Learning Systems

Anonymous Submission

Abstract

Machine-learning (ML) ecosystems that support todays organizations all too often fail to implement rigorous data protection, exposing user data to hackers and privacy-transgressing employees. We posit that one reason is the insufficiency of existing protection mechanisms and abstractions – such as access control lists on files, tables, or streams – to sustain the data access and sharing patterns required in these systems. We propose a *private transferrable knowledge* (PTK) abstraction, a new unit for rigorous data protection, sharing, and retention in ML ecosystems. PTKs are feature models trained over long-term historical data and made differentially private to protect the training data. Using plausible corporate scenarios crafted around public datasets, we demonstrate the value of PTKs as protected units of data sharing and retention. Around PTKs, we build *Sage*, a system that creates, maintains, and shares multiple PTKs on behalf of a wide range of ML workloads. A critical challenge in Sage is to enforce a global privacy semantic for PTKs at scale and while minimizing exposure of user data through its own internal state. We resolve this challenge with a novel double-resource allocation approach to privacy budget management in DP systems.

1 Introduction

Data-rich machine learning (ML) ecosystems arising in today's companies, governments, and organizations all too often fail to implement rigorous data protection. This failure exposes raw user data to unintentional, unwanted, or improper uses of privilege by both employee and external attackers [22, 28]. Take, for example, the "data lake" architecture now emerging in many ML ecosystems [3]. user data from a company's multiple products is pooled and integrated into a single, vast repository, which archives it for indefinite time periods and makes it accessible to all corporate data engineers who might have some use for it. Such architectures highlight the need for abstractions that protect, control sharing of, and retain sensitive information.

Traditional protection and sharing abstractions – such as files, directories, database tables, and views – were designed for traditional software, where it was very clear what data was needed to implement specific functions. For example, a "traditional" social networking application might consist of several services, each requiring a different subset of user data to do its job. The authentication service might need access to the user account database; the social networking service to users' friends lists and posts but not to the account database or credit card information; and an in-app product purchase service to credit card but not social graph or account information. Companies could store these different data types in separate database tables and enable access to various services and teams strictly on a needs basis.

Emerging data-hungry ML applications, often built on data lake architectures [35], make no such clear distinctions. Thus, data collected to improve a post recommendation service, including posts previously read/liked by each user, are considered relevant not only for that service but for others aimed at friend recommendations, ad targeting, and potentially bot detection. Similarly, data collected for ad targeting, such as ad clicks or visited Web pages, are deemed relevant both for post and friend recommendation services and for bot detection. In some cases, the data may ultimately prove useless. However, knowing whether or not it is useful often requires its use – and therefore access to its files or tables – for experimentation and possible evaluation in production.

The benefits of leveraging existing corporate data must, of course, be weighed against the growing costs of privacy vulnerabilities on a company-specific basis. We posit that many companies have adopted casual protection principles due to the lack of appropriate and efficient data protection, sharing, and retention abstractions for ML workloads. The development and use of new abstractions for these purposes can better synchronize cost/benefit tradeoffs with specific departmental or divisional needs and with corporate data use policies.

In many cases, we observe that what is needed in data sharing scenarios is not access to raw data, but access to "knowledge" drawn from that data, such as historical statistics, well-engineered features, or *models* trained on the data. A friend recommendation service may find more benefit in incorporating user embeddings already trained and curated over the post service's data than from the raw posting activity of its users. Embeddings – which in this case map each user to a low-dimensional space that captures user similarities in terms of posts read, written, and liked – can be more easily incorporated into new ML tasks than raw data, which is highly dimensional and messy. User embeddings are now shared across teams at large companies, including Twitter [48].

Given these considerations, we propose a new abstraction, *private transferrable knowledge* (PTK), a unique approach to protecting, sharing, and retaining data in ML

systems. PTKs are machine learning models trained over historical data and made differentially private (DP) to protect training data confidentiality. They are intended to be protected units of retention by teams needing raw access to personal data streams, and as units of sharing with other teams that do not require raw data access. As units of retention, PTKs are trained over extended periods of time and thus encode information about the past; because they are private, they can be retained for long periods of time with no concerns about exposure. When incorporated into predictive tasks, PTKs can improve the rate of learning thereby allowing these tasks to train on recent windows of raw data without loss of accuracy. As units of sharing, PTKs are trained over one data stream and shared with predictive tasks to increase their accuracy without concerns of cross-team data exposure.

While in theory any DP ML model can be considered a PTK, our design focuses on the specific case of *feature models*, such as user embeddings, historical statistics, and clustering information. Our rationale for this focus is two-fold. First, well-engineered features, though complex and time consuming to build, are critical components of any ML ecosystem. Appropriate features make model training and optimization easier and less data intensive because they already capture significant historical characteristics. Second, increasing evidence shows that sharing well-engineered features across teams is common practice in today's data-driven companies. Uber and Instacart have developed feature stores that help development teams share tens of thousands of curated features [35, 4]. We believe that elevating feature models to the rank of protection abstraction is a valuable step toward more rigorous protection practices in ML systems.

This paper makes the following contributions to the protection and control of data sharing in ML systems:

1. PTKs, a novel DP feature model abstraction that can be used as a protected unit for long-term retention and cross-team sharing in an ML ecosystem.
2. The Sage PTK Store, a minimal-exposure DP ML system that manages, trains, and makes available multiple PTKs for an organization's data streams. Conceptually, Sage differs from feature stores in today's companies in two major ways: (1) it accepts only feature learning procedures that preserve DP guarantees, directly tapped from the enormous body of existing work on DP learning algorithms [9, 24], and (2) it enforces tight global privacy semantics for PTKs externally while minimizing raw data exposure through its internal data structures.
3. The first double-resource allocation approach for DP privacy budget management. It assigns not only privacy budgets (as is customary) but also training samples to achieve a tight global privacy semantic with many PTKs while minimizing the total number of samples needed to achieve acceptable per-model accuracy. Our approach relies on a new method to estimate sample complexity bounds for DP models and a new integer program for resource allocation.

We evaluate PTKs and Sage with three corporate-inspired scenarios on six publicly available datasets. Across scenarios, using PTKs instead of unprotected features affects predictive model accuracy by 0-4.3%. Thus, PTKs are useful and practical abstractions that can be implemented in a data-rigorous way.

2 Example Scenarios

Scenario 1 (Retention): An online advertising startup collects various user activity streams and uses them to target ads. As a market differentiator, the company would like to position itself as privacy-conscious by imposing tight data retention policies on their user data. They observe that with good features trained over long periods of time, their models require much less data to converge. They want to safely retain these features for extended periods of time while limiting the timespan of raw data they use to train predictive models.

Scenario 2 (Sharing): A transportation company is already operating a large fleet of taxis in a city. A new division of that company would like to launch a bike sharing system in the same city. The bike share division would like to use the ride information from the taxi operations to bootstrap forecasting bike demand in different areas of the city, because they expect that the taxi and bike services will be used for similar trips. To limit data exposure, they wish to share the taxi information privately. The bike division expects the information to boost performance of its forecast while the bike datasets are small.

Scenario 3 (Envisioned Sage Use): Company X uses Sage as the de facto feature store in their ecosystem. Teams with access to certain data streams design good features over these streams for their own tasks and publish them as PTKs over Sage. Internally, they use the PTKs to improve their models while keeping tabs on user data exposure through their services. Other teams look to Sage to find PTKs that may be relevant for their own tasks, evaluate whether they improve their tasks, and if so, incorporate them in production. Sometimes teams coordinate to develop appropriate PTKs for multiple tasks.

3 Goals and Background

Our goal is to develop a *strong-semantic data sharing and retention abstraction* that allows principled tuning of access to user data in ML ecosystems. The abstraction should enable organizations that currently over-expose user data – through wide-access policies or long retention periods – to limit that exposure without sacrificing workload accuracy. It should also enable companies that currently over-constrain data access – through siloed ar-

chitectures or short retention periods – to increase workload accuracy without increasing data exposure.

Specifically, we formulate three requirements for the abstraction and the system that implements it:

R1: *Suitable abstraction for ML workloads.* The protection abstraction must naturally fit into common patterns in ML ecosystems and support varied data sharing and retention use cases such as those in the preceding section. The abstraction must also support a wide range of predictive tasks (including regression and classification) and ML models.

R2: *Limited impact on workload accuracy.* When used to reduce exposure, the abstraction should result in minimal loss of accuracy for predictive tasks compared to the best alternative without using it. When used to improve accuracy, we aim for notable improvement to justify computational overhead.

R3: *Rigorous-semantic and minimal-exposure system.* The system that implements this abstraction (Sage) should enforce rigorous protection semantic for it. Moreover, Sage should itself be a model of rigorous data management and minimize the amount of data it exposes through its internal structures at all times.

3.1 Threat Model

We consider ecosystems where streams of user data are being collected for use in various predictive ML tasks. We assume that each stream has a set of *primary tasks* that require access to the stream's raw data to function. We additionally assume the existence of other tasks that do not need access to the raw data in that stream but that would benefit from statistics or features learned from the stream in a multitask or transfer learning setting (defined in §3.3). We call these *transfer tasks* and assume they have their own raw data.

We are concerned with two classes of adversaries who have or gain access to the company's internal state and data stores. First are *abusive employees*, who leverage their privilege within the company to learn about friends' or family members' interactions with the company's products. These adversaries cannot intrude past traditional access controls to escalate their privilege, but they can continuously monitor for nefarious purposes any data or state to which they have legitimate access. With the notable exception of the Sage administrators, all employees constitute a risk. Second are *intruders*, who have no legitimate access to the company's internal state, but who manage to break into its compute resources to gain access. For example, they might hack an employee's laptop and gain access to any state accessible by the employee. Alternatively, they might identify a vulnerability in a running service (such as a buffer overflow or a heartbleed-like vulnerability) and retrieve that service's state from memory or stable storage. Intruders can, at worst, access arbitrary state within one or more compromised services, appear at any time and without prior notice to the organization, and appear multiple times within the organization's lifetime. However, we assume that intruders are not continuously present in the company nor able to continuously monitor the company's external predictions to its users. Both classes of attacks are highly relevant, as evidenced by numerous media reports exemplifying each: [26, 44], [22, 28]

We seek an abstraction that will let companies reduce exposure of their user data streams to both types of attackers. First, we want to protect the sharing of features from a source data stream to its *transfer tasks* without increasing the data's exposure to the potentially abusive employees in charge of these tasks or to the compromisable services running them. Sharing raw data constitutes exposure, as does sharing *any state* computed from the data (including the features) and not protected with a differential privacy or other cryptographic guarantee. Prior research has shown that even the most innocuous-looking aggregate statistic or parameters of ML models, can reveal a significant amount of information about individual examples in their training sets [49]. Second, we want to reduce the amount of raw user data exposed at any time to intruders through that data's *primary tasks*. A rigorous approach to such reduction is to limit the data's retention period in its primary tasks.

3.2 Candidate Approaches

A protection abstraction has two parts: (1) a protection mechanism (e.g., access control lists in traditional systems), and (2) a unit for protection (e.g., files or tables in traditional systems). For ML systems, we base our protection abstraction on *differential privacy* (DP) applied to *feature models*. While other mechanisms and units exist, we argue that they are either not well suited to ML workloads or will likely need to be combined with our abstraction to sufficiently protect against our adversaries.

An alternative protection mechanism applies homomorphic encryption (HE) [27, 33] or secure multi-party computation (MPC) [42, 41] to traditional protection units, such as files or streams. These let data engineers train models "blindly" on one another's data streams. Putting aside the significant performance concerns of running HE or MPC mechanisms, a key limitation of using encryption as the sole protection mechanism in ML systems is its lack of adequate protection: trained models, whose decryption is usually assumed for prediction, can leak training data information [23, 49]. Fixing this requires either an additional MPC protocol with the end users [15], which adds to the performance concern, or combination with DP. Thus, we view DP as a crucial component of any protection abstraction.

As an alternative unit of protection, we could apply DP at the level of *SQL queries*. For example, consider a company that requires all access to raw data to go through

a DP SQL interface, such as PINQ [30] or FLEX [39]. This approach works well for workloads easily written on SQL, such as simple aggregates or learning algorithms that can run efficiently on statistical queries [14]. However, implementing broad classes of learning algorithms, including those based on stochastic gradient descent, in SQL is both challenging and inefficient. We believe that the SQL interface is too low-level an abstraction at which to enforce protection in ML workloads. We therefore elevate the level of abstraction to *machine learning models*, and specifically, to *feature models*.

3.3 Differential Privacy Background

DP offers a well-defined semantic for preventing leakage of individual records in a dataset through the output of a computation over that dataset. It works by adding randomness into the computation that "hides" details of individual records. A randomized algorithm A that takes as input a dataset D and outputs a value in a space B is said to satisfy ϵ-*DP at record level* if, for any datasets D and D' differing in at most one record, and for any subset of possible outputs $S \subseteq B$, we have: $P(A(d) \in S) \leq e^{\epsilon} P(A(d') \in S)$. Here, $\epsilon > 0$, called the *privacy budget*, is a parameter that quantifies the strength of the privacy guarantee (lower is better); $\epsilon \leq 1$ is generally considered to be adequate protection.

Three important properties of DP are: (1) *Post-processing resilience*: any computation applied on the output of an ϵ-DP algorithm remains ϵ-DP [20]. (2) *Sequential composition theorem*: if the same dataset is used as input for two DP algorithms, ϵ_1-DP A_1 and ϵ_2-DP A_2, then the combined privacy semantic is $\epsilon_1 + \epsilon_2$-DP [39]. (3) *Parallel composition theorem*: if two DP algorithms, ϵ_1-DP A_1 and ϵ_2-DP A_2, use as input two disjoint datasets (e.g., two non-overlapping windows of the same stream, two different data streams, or two randomly selected disjoint subsets of a dataset), the combined privacy semantic is $max(\epsilon_1, \epsilon_2)$-DP [39].

3.4 Scope

The sharing use cases we aim to support with our abstraction correspond to well-understood and common multitask and transfer learning scenarios. Multitask [52] and transfer learning [45] exploit commonalities across different learning tasks to improve the amount of information needed for successful learning. For instance, if good solutions for two related tasks use the same set of features, then effort spent identifying the features when solving one task can be re-used to make solving the other task easier (i.e., require fewer training data).

4 PTK Abstraction

We propose *private transferrable knowledge* (PTK), a new protection abstraction specifically designed for ML systems. Its unit of protection is the *feature model*; its protection mechanism is *differential privacy* (DP). We believe that this abstraction, together with the system that implements it, meets the three requirements defined in §3. First, feature models are already being used as units for data sharing in today's ML ecosystems [35]. This suggests that they will also serve as natural and suitable units for protected data sharing in these systems (requirement **R1**). Second, the literature is rife with DP implementations of most popular ML algorithms [37, 9] and increasing experimental evidence suggests that DP is amenable to ML [34, 31]. This suggests that DP can be applied to feature models with limited accuracy overhead (requirement **R2**). Finally, §5 shows how to enforce a global privacy semantic for PTKs while minimizing the data used to train them (requirement **R3**).

```
struct ptk: id, model_state, config, τ,
   oldest_contributing_window, newest_contributing_window

// PTK developer API:
ptk.train_dp(trainset_iter, ε_train,
    previous_model_state, previous_config) → overwrite
ptk.eval(testset_iter) → PTK-specific evaluation metric

// PTK user function (not part of API):
featurize(targetset_iter, ptk[]) → featurizedset_iter

// Sage API:
register_ptk(ptk_train_dp_fn, ptk_eval_fn, train_frequency)
    → ptk_id OR REJECT
deregister_ptk(ptk_id)
update_ptk_config(ptk_id, new_config)
update_ptk_performance_target(ptk_id, new_τ)
subscribe_to_ptk(ptk_id, rpc_endpoint)
notify_new_ptk_state(ptk_id)
```

Fig. 1: **PTK API.**

4.1 API

Fig. 1 shows the PTK API, which is used by PTK developers and users. **PTK Developer:** The developer first defines a procedure to learn useful features from an accessible data stream and then turns that procedure into a PTK by implementing the PTK developer API (described below). The developer then identifies a reasonable performance target τ for the PTK, which Sage attempts to honor when it assigns privacy budgets and training samples to PTKs. The final step is registering the PTK with Sage (`register_ptk`) and giving Sage access to its source data stream. Sage assigns a unique ID to the PTK, but can also reject the PTK if it lacks sufficient privacy and data resources to fulfill its performance target (§5).

PTK User: The user of a PTK can be either the same entity that developed it (primary task) or a separate entity that uses it in a transfer learning setting (transfer task). To use a previously registered PTK, the user subscribes to it by calling `subscribe_to_ptk`, specifying the PTK's ID and an RPC endpoint. Sage trains the PTK periodically, stores it in a widely accessible *PTK lake*, and notifies the user of new model states available for it. The user retrieves those states from the PTK lake and incorporates them into predictive models. Our abstraction imposes no API for PTK use; usually, a user will implement a `featurize` function that transforms the dataset based on one or more PTKs.

	PTK	Type	ptk_train_dp	ptk_eval	Used in
1	**Count featurization**	S	DP contingency tables [11]	error from true value	Scenario 1
2	**Tree featurization**	S	Large margin mechanism[17]	loss on a holdout set	Scenario 2
3	**User embeddings**	S	DP Poisson factorization [25]	loss on holdout set	Scenario 3
4	**Covariance matrix**	U	Analyze gauss [21]	error from true value	Scenario 3
5	**Historical statistics**	U	Aggregates w/ Laplace noise [20]	error from true value	Scenario 3
6	**Clustering**	U	DP Lloyd algorithm [51]	loss on training set	Scenario 3

Tab. 1: **Implemented PTKs.** Type: Supervised/Unsupervised. Core methods used to implement the PTK API. Scenario where we used the PTK.

PTK Developer API: We designed the PTK API for simplicity and naturalness. It consists of two functions. `ptk.train_dp` is a DP version of the developer's training procedure. It takes a *training set* and a privacy budget, ϵ_{train}, and must satisfy ϵ_{train}-DP. We train PTKs periodically on $\mathcal{T}$-sized time windows, where $\mathcal{T}$ is the exposure window that Sage is willing to tolerate for the PTK's data stream (§5). We support continuous PTK training across windows by supplying `ptk.train_dp` with the previous state of the model. `ptk.eval` is a PTK evaluation procedure that takes as input a testing set and returns a PTK-specific loss metric. The `eval` function need not satisfy DP.

4.2 Example PTKs

We implemented *ptklib*, a library of PTKs for popular feature learning algorithms. We use a recent feature engineering textbook [54] to prioritize algorithms for implementation. Tab. 1 shows the PTKs implemented thus far, the type of algorithm (supervised/unsupervised), and the methods we used to implement the developer API for each PTK. For most PTKs, we used: known DP versions of training algorithms for `ptk.train_dp` and sensible performance metrics for `ptk.eval`. We exemplify our implementation of three PTKs particularly relevant for our evaluation of the two scenarios in §2.

Count Featurization PTK. Count featurization is a popular technique for featurizing high cardinality categorical variables, such as user identifiers and IPs, when training classification models [50, 54]. The technique replaces each value of the feature vector with the number of times that feature value has been observed with each label and the conditional probability of each label given that feature value. This leads to dramatic dimensionality reduction over standard one-hot encoding and enables more efficient learning. In our context, this means that the PTK user may be able to learn related labels using less data by featurizing the data with DP counts and conditional probabilities trained on historical data streams. In an instantiation of Scenario 1, §6.2 shows how to use this PTK to reduce data retention in primary tasks.

PTK API: `ptk.train_dp` computes contingency tables of each feature with each label. It makes the tables DP by initializing their cells with random draws from a Laplace distribution [11], scaled by ϵ_{train}. `ptk.eval` returns the error from the true value.

Tree Featurization PTK. Tree featurization is a technique to develop informative nonlinear features by constructing a forest of decision trees. Each root-to-leaf path in a decision tree corresponds to a conjunctive (and hence nonlinear) predicate of the input. Hence, root-to-leaf paths capture informative nonlinear interactions between the features. If these paths are then used to featurize a related task's data set, then they can improve that task's performance especially when the task lacks sufficient data to derive these interactions on its own. The technique is being used at Facebook [29] and has a scikit-learn implementation. In an instantiation of Scenario 2, §6.3 uses this PTK to bootstrap a new transfer task that lacks sufficient data of its own.

PTK API: `ptk.train_dp` computes DP a random forest for each label. We use a known approach to making decision trees private [24]. On every node, we: (1) choose a random subset of features to split on; (2) compute the predictive utility of all splits over these features using gini impurity for classification and mean squared error for regression; and (3) privately choose a version of the best split using the least margin mechanism [17]. We split the ϵ_{train} privacy budget equally across trees in the forest, and within a tree across its layers, such that overall, the forest satisfies ϵ_{train}-DP. `ptk.eval` returns the mean squared error or gini impurity on the testset.

User Embeddings PTK. Embeddings are a broad class of techniques used to create low dimensional representations of high dimensional features, such as user ids or a word from a word dictionary. These low dimensional representations expose the underlying similarities between elements of the high dimensional feature, such as users' topics of interest, and thus enable efficient learning on small amounts of data. For this reason, they are wildly used in ML and often shared across teams at big companies [47]. Our User Embeddings PTK leverages Bayesian hierarchical Poisson matrix factorization [25] to create a user embedding, also called a latent representation, from a dataset of interactions between users and items (e.g. movie ratings or music listened to). In an instantiation of Scenario 3, §6.4 uses the User Embedding PTK of a movie recommendation application to improve the ratings prediction on books and music.

PTK API: `ptk.train_dp` computes a DP version of hierarchical Poisson matrix factorization [25]. This computation requires two changes to the original algorithm. First, we use stochastic Variational Inference [13] as a training procedure, allowing updates from small batches of data that give better DP guarantees to each up-

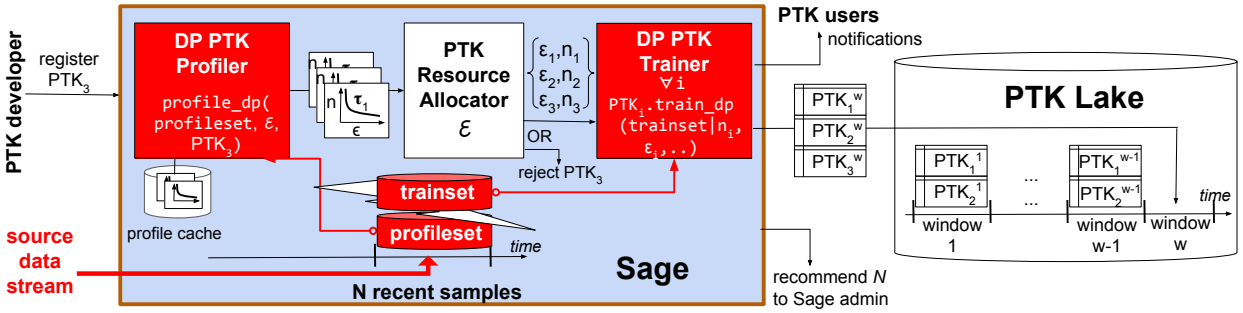


Fig. 2: **Sage Architecture.** PTK_i: PTK object; PTK_i^j: the state of PTK_i at time window j; $trainset|_{n_i}$: n_i-sized partition of trainset.

date. Second, we use the Moment Accountant from [10] to compute tight bounds on the privacy loss, and make the DP training procedure manageable. `ptk.eval` returns the negative log likelihood of the testing set under the Bayesian hierarchical Poisson matrix factorization model. Since a good fit maximizes log likelihood, the negative log likelihood is a loss.

5 Sage

To realize the PTK abstraction, we developed the *Sage PTK store*, a minimal-exposure system that creates, trains, and shares PTKs on a company's data streams. The unique aspect of its design is its effort to minimize at all times the size and timespan of the user data exposed through its own internal state (requirement **R3** in §3).

5.1 Architecture

Fig. 2 shows Sage's architecture. It consists of two top-level components: *the Sage service*, which ingests several of the company's user streams and trains PTKs on them periodically; and a *PTK lake* that stores the trained PTKs (including the models' parameters and configurations) and makes them widely accessible to any engineer, team, or service who might have some use for them. To train the PTKs, Sage keeps a tumbling window of recent data from each stream, called the *training window*. We denote $\mathcal{T}$ the timespan of the training window and $\mathcal{N}$ its size. Different PTKs can be computed at different intervals of time, but for simplicity we will assume here that they are trained once per training window.

Protection Semantic. To protect user data against adversaries listed in §3.1, Sage enforces a two-fold protection semantic. First, because PTKs are widely accessible in the PTK lake, they can be monitored by abusive employees or hackers with access to the lake. Each PTK enforces ϵ_{train}-DP, hence for small ϵ_{train} the leakage is small from individual PTKs. However, across PTKs in the same training window the leakage is additive (§3.3). To bound the leakage across PTKs, Sage enforces a *global privacy semantic*, $\boldsymbol{\mathcal{E}}$-DP, for $\mathcal{E}$ configured by the Sage administrator (default $\mathcal{E} = 1$).

A second point of vulnerability is the ingestion of multiple data streams by Sage. While the Sage administrator is trusted to not abuse this data, the service itself can be compromised by intruders. To limit exposure, Sage minimizes at all times the timespan of user data that can be leaked through its internal state. Specifically, it: (1) enforces that at any time, any non-DP state only depends on data in the training window, which has a timespan of $\mathcal{T}$ and (2) helps tune $\mathcal{T}$ to the minimal value needed to gather sufficient samples to reach per-PTK performance targets. We call this the *minimal exposure semantic*, parameterize it by $\mathcal{T}$, and think of it as Sage's own data retention policy.

Challenge. Enforcing a global $\mathcal{E}$-DP guarantee across multiple queries/data uses is a known challenge that has been considered in the literature [53, 40]. However, minimal-exposure semantic is unique to our system and introduces new challenges. Recall that each PTK_i comes with a performance target, τ_i, configured by its creator. If we can generically and rigorously estimate the sample size required for each PTK_i to reach its performance target, then we could assess how much data Sage needs to train all PTKs jointly. However, this sample size n_i needed by PTK_i is generally a complex and unknown function of the target performance τ_i and the allocated privacy budget ϵ_i. Herein lie two challenges: (1) how to rigorously and generically estimate the sample size needed achieve a given PTK performance and (2) how to allocate privacy budgets and data samples across PTK to minimize data use under the privacy constraint.

Approach. Our approach has two steps. First, we *profile* each PTK in isolation, as it is registered with Sage, in terms of how its performance is impacted by various values of ϵ_i, n_i. For that we develop *the first generic complexity bound estimation algorithm for DP models* that fit the empirical risk minimization framework. Second, we *find an allocation* (ϵ_i, n_i) for all i that preserves the global privacy guarantee ($\mathcal{E}$) and minimizes the total amount of data needed for training ($\mathcal{N}$) while meeting the PTKs' performance targets. For that we develop *the first double-resource allocation formulation for DP systems* that simultaneously assigns privacy budgets and samples to DP models.

Fig. 2 reflects the Sage architectural components that implement this procedure: *DP PTK Profiler*, *PTK Resource Allocator*, and *DP PTK Trainer*. At every win-

dow, we partition the $\mathcal{T}$-window of recent data in two: a *profileset* and a *trainset*. The *PTK Profiler* uses the profileset to profile a small set of new or not recently profiled PTKs (PTK_3 in the figure). The profiles, which are made DP, are saved in a profile cache for future reuse. The *PTK Resource Allocator* uses the profiles of all PTKs registered in the system to determine an allocation (ei, ni) for each PTK_i that satisfies the conditions in the preceding paragraph. It is possible that the allocator cannot find such an allocation, in which case it will REJECT the new PTK (PTK_3) or otherwise announce the Sage administrator that a sound allocation does not exist under the current PTK mix, and revert to its previous allocation plan. The *PTK Trainer* uses the allocation to train the PTKs on the *trainset*. Finally, Sage automatically determines a proper value for $\mathcal{N}$, to support both PTK profiling and training. It recommends the value to the administrator, who can decide to update the $\mathcal{T}$ parameter of the minimal-exposure semantic.

5.2 DP PTK Profiler

The DP PTK Profiler uses a held-out data set (profileset) to estimate the number of samples needed by the PTK to reach a particular performance target, τ, as a function of the privacy budget it is allocated for training, all the while guaranteeing differential privacy with respect to the profileset during this process. In principle, a PTK developer could provide a profiling procedure for her PTK. In practice, profiling DP models is challenging and readily available in the literature. So we develop *a generic method for rigorously estimating the sample complexity of a DP predictive model*.

Problem Formulation. We focus on PTKs based on finding a predictor f that approximately minimizes a *risk* $\mathcal{R}(f) := \mathbb{E}[\ell(f, X)]$, i.e., the expected loss of f on a random example $X \sim P$ drawn from a distribution P over $\mathcal{X}$; here, P is the profileset in Sage, and $\ell(f, x)$ is the $[0, 1]$-valued loss of f on the example $x \in \mathcal{X}$. Let $\mathcal{F}$ be a fixed *function class* and $f^* := \arg\min_{f\in\mathcal{F}} \mathcal{R}(f)$ be the predictor of smallest risk. (This formulation covers both supervised and unsupervised learning.) When $f_n \in \mathcal{F}$ is chosen to minimize *empirical risk* $\mathcal{R}_n(f_n) := \frac{1}{n}\sum_{i=1}^n \ell(f_n, x_i)$ on an iid sample $(x_i)_{i=1}^n$ from P, the empirical risk $\mathcal{R}_n(f_n)$ under-estimates $\mathcal{R}(f_n)$ by amount $\delta_{\mathcal{F},P,n}$ that typically (i.e., with high probability over the draw of $(x_i)_{i=1}^n$) depends on the sample size n, a "capacity" measure of $\mathcal{F}$, and interaction of $\mathcal{F}$ with P [12].

For the PTKs of interest, the functional dependence of $\delta_{\mathcal{F},P,n}$ on n is, up to log factors, $O(n^{-1/2})$. This means there is some sample size $n_{\mathcal{F},P,\tau}$ such that if $n \geq n_{\mathcal{F},P,\tau}$, then $\delta_{\mathcal{F},P,n} \leq \tau$. However, the dependence of $\delta_{\mathcal{F},P,n}$ on $\mathcal{F}$ and P is more nuanced; known *a priori* upper-bounds on $\delta_{\mathcal{F},P,n}$ are notoriously loose. So, it is not generally possible to "invert" $\delta_{\mathcal{F},P,n}$ to derive a useful formula for $n_{\mathcal{F},P,\tau}$. The situation is similar for the risk of an $\mathcal{E}$-DP approximation f'_n, such as that computed by the Exponential Mechanism on the empirical risk functional: the risk $\mathcal{R}(f'_n)$ may be larger than $\mathcal{R}(f_n)$ by an amount $\delta_{\mathcal{F},P,n,\mathcal{E}}$ that is $O((\mathcal{E}n)^{-1})$ in terms of $\mathcal{E}$ and n, but dependence on $\mathcal{F}$ and P is more complex [32, 16].

Our goal is to use a stream of iid data from P to approximate the minimum sample size $n_{\mathcal{F},P,\tau,\mathcal{E}}$ such that $\delta_{\mathcal{F},P,n} + \delta_{\mathcal{F},P,n,\mathcal{E}} \leq \tau$. Our current methodology is limited in that we achieve this goal only when $\mathcal{R}^* := \min_{f\in\mathcal{F}} \mathcal{R}(f) \leq \tau$: we are able to find $\hat{f} \in \mathcal{F}$ (using an $\mathcal{E}$-DP algorithm) on the basis of a random sample of size approximately $n_{\mathcal{F},P,\tau,\mathcal{E}}$ such that $\mathcal{R}(\hat{f}) \leq \mathcal{R}^* + \tau \leq 2\tau$. Because of this limitation, we would also like to be able to detect when $\mathcal{R}^* > \tau$. However, it is impossible to distinguish between $\mathcal{R}^* = \tau$ and $\mathcal{R}^* = \tau + \delta$ for very small $\delta > 0$ without an exceedingly large sample size. We consider a relaxed goal of correctly declaring that $\mathcal{R}^* > \tau$ only when $\mathcal{R}^* > 2\tau$, which is possible with sample size $\approx n_{\mathcal{F},P,\tau,\mathcal{E}}$. If $\tau < \mathcal{R}^* \leq 2\tau$, either finding is permitted.

DP Profiling Algorithm. We assume access to an *empirical risk minimization* (ERM) algorithm for $\mathcal{F}$: given data $(x_i)_{i=1}^n$ from $\mathcal{X}$, the ERM algorithm returns a minimizer f_n of the empirical risk $\mathcal{R}_n$ (based on $(x_i)_{i=1}^n$) over all $f \in \mathcal{F}$. We also assume access to an $\mathcal{E}$-DP algorithm for $\mathcal{F}$; for simplicity, we assume that the algorithm is the Exponential Mechanism (EM) on the empirical risk functional for data $(x_i)_{i=1}^n$ [38, 32].

Our profiling algorithm, shown in Fig. 3, uses the stream of iid data for two purposes. The first use is for running ERM and EM on iid samples S_n of increasing sizes n. One property used is that with high probability over the draws of the S_n, we have $|\mathcal{R}_n(f^*) - \mathcal{R}^*| \leq \delta_n^* := \sqrt{\ln(100T)/(2n)}$ for each n considered in the algorithm, where $T := \lfloor \log_2 n_{\max} \rfloor$. The second use is to form a validation set $V_m := (x'_i)_{i=1}^m$ of size $m := 2(\tau^{-2} + (\epsilon\tau)^{-1}) \ln(100T)$. For any collection of T predictors $(f_j)_{j=1}^T$, with high probability over the draw of V_m, the empirical risk $\mathcal{R}_m^{\text{val}}$ on $(x'_i)_{i=1}^m$ (which we call *validation risk*, to distinguish it from $\mathcal{R}_n$) of each f_j satisfies $|\mathcal{R}_m^{\text{val}}(f_j) + Z_j - \mathcal{R}(f_j)| \leq \tau/4$, where $Z_1, \ldots, Z_T \sim_{\text{iid}} \text{Lap}(1/(\epsilon m))$.

The main idea of the algorithm is to compute *differentially-private upper and lower bounds on* $\mathcal{R}^*$ based on iid samples of geometrically increasing sizes. Importantly, these bounds do not depend on the unknown quantities $\delta_{\mathcal{F},P,n}$ and $\delta_{\mathcal{F},P,n,\mathcal{E}}$; these quantities are only needed in the analysis of the algorithm.

Guarantees. As soon as n is larger than $n_{\mathcal{F},P,\tau/4,\mathcal{E}} + O(\log(T)/(\epsilon\tau))$, the width of the interval $U_n - L_n$ will, with high probability, be smaller than $\tau/2$, upon which we will be able to either return (n, f'_n) or declare "$\mathcal{R}^* > \tau$". This is typically within a small constant factor of the target sample size $n_{\mathcal{F},P,\tau,\mathcal{E}}$.

- Given: privacy parameter ϵ, maximum sample size $n_{\max}$.
- Draw iid sample V_m from P of size m.
- For $j = 0, 1, \ldots, \lfloor \log_2 n_{\max} \rfloor$:
  1. $n := 2^j$.
  2. Draw iid sample S_n from P of size n.
  3. $f_n := \mathrm{ERM}(S_n)$, and $f'_n := \mathrm{EM}(S_n)$.
  4. $L_n := \mathcal{R}_n(f_n) - \delta^*_n + Z^L_j - \frac{\ln(2T)}{\epsilon n}$, where $Z^L_j \sim \mathrm{Lap}(\frac{1}{\epsilon n})$.
  5. $U_n := \mathcal{R}^{\mathrm{val}}_m(f'_n) + Z^U_j + \frac{\tau}{4}$, where $Z^U_j \sim \mathrm{Lap}(\frac{1}{\epsilon m})$.
  6. If $U_n \leq 2\tau$, then return (n, f'_n).
  7. If $L_n > \tau$, then return "$\mathcal{R}^* > \tau$".

Fig. 3: **Profiler Algorithm.** (m, T, and δ^*_n defined in main text.)

given	$\mathcal{E}$:	global privacy parameter
	K :	number of partitions
	ϵ_j :	profiled privacy budgets
	n_{ij} :	samples needed by PTK_i to reach performance τ_i with ϵ_j-DP
variables	x_{ijk} =	$\mathbb{1}\{$assign PTK_i to part k with $\epsilon_j\}$
minimize	$\sum_k \max_i \sum_j x_{ijk} n_{ij}$	
subject to	$\forall k. \sum_i \sum_j x_{ijk}\epsilon_j \leq \mathcal{E}$	privacy constraint
	$\forall i. \sum_j \sum_k x_{ijk} = 1$	assign constraint

Fig. 4: **Double-Resource Allocation Integer Linear Program.**

This guarantee assumes that the profiler uses ERM and EM algorithms for $\mathcal{F}$. Neither assumption is generally satisfied except in restricted settings. Instead, predictors are often picked using heuristic algorithms (e.g., local search, greedy methods) whose DP variants do not always behave like EM. However, regarding these algorithms as ERM and EM is a plausible model for how the heuristics behave in practice; e.g., local search heuristics are commonly observed to find parameters of neural nets that minimize empirical risk.

When neither assumption holds, then the profiler may fail to return (n, f_n) in cases where it should have been possible (using ERM and EM); it may also falsely return "$\mathcal{R}^* > \tau$." However, any (n, f_n) returned by the profiler is a witness that the ϵ-DP training algorithm is able to find a predictor with risk at most 2τ.

Finally, the overall procedure guarantees $\max\{\mathcal{E} + \epsilon, J\epsilon\}$-DP, where J is the total number of iterations (and $J \leq \log_2 n_{\max}$). It is straightforward to modify the procedure to guarantee $\mathcal{E}$-DP by using different iid samples in Steps 3 and 4, and appropriately setting ϵ.

5.3 PTK Resource Allocator

The PTK Resource Allocator takes in DP profiles for all PTKs and allocates privacy budgets (ϵ_i) and number of training samples (n_i) for each PTK, with two requirements: (1) the global privacy semantic $\mathcal{E}$-DP is satisfied across PTKs and (2) the total number of samples allocated to the PTKs is minimized.

Problem Formulation. We formulate this problem as an integer linear program (ILP). Denote ϵ_j the values with which we profiled the PTKs, and n_{ij} the number of samples needed by PTK_i to reach its performance target with ϵ_j-DP. The ILP is over the assignment variables $x_{ij} = \mathbb{1}\{$assign PTK_i privacy budget $\epsilon_j\}$ is $\min_{(x_{ij})} \sum_i x_{ij} n_{ij}$ subject to $\sum_{ij} x_{ij}\epsilon_j \leq \mathcal{E}$, $\sum_j x_{ij} = 1$, and $x_{ij} \in \{0, 1\}$.

One issue is that with large numbers of PTKs and small privacy budget, the ILP may have no solutions. This is a fundamental challenge in DP systems on static datasets: the privacy budget is *not* a scalable resource. We observe that in our streaming case, there is another, more scalable resource – the sample size – which we could trade to allow solutions even with many more PTKs. Due to the parallel composition theorem of DP (§3.3), if we partitioned the data, say in K disjoint parts, and trained groups of PTKs separately in each partition, then the combined privacy semantic will be the maximum of the privacy semantic achieved in each partition. However, the total amount of data used in that case would be the sum of the samples needed by the PTKs in each partition. By letting the resource allocator use more data than what is strictly needed by the most data-intensive PTK, this new formulation lets it find allocations when strictly splitting the privacy budget is not possible.

Resource Allocation Algorithm. Fig. 4 shows the integer program formulation we use in Sage. Given a number of partitions, K, it assigns PTKs to "partitions" and privacy budgets such that: (1) within each partition, the total privacy budget is $< \mathcal{E}$ (privacy constraint) and (2) every PTK is assigned to some partition (assignment constraint). The objective of the algorithm is to minimize the sum of the number of samples needed by the PTKs in each partition (objective). The program can be proven NP-hard through a reduction from the NP-complete `PARTITION` problem, but in our experience, the Gurobi solver finds a solution in seconds for up to 50 PTKs. Note that the initial formulation we expressed is equivalent to $K = 1$ in the Sage formulation. In our implementation, we run with increasing values of K until we find a solution.

Guarantees. Assuming that all PTKs can reach their target semantic with $\mathcal{E}$-DP, then our algorithm is guaranteed to have a solution: assigning each PTK in a different partition. However, the total number of samples may be extremely large, and hence the exposure risk of maintaining such large datasets may be inadmissible. We envision an administrator configuring Sage with a maximum number of samples N_{max} that it cannot cross. Sage will try to accommodate the given PTK workload, at all times with the minimal exposure within N_{max}, and if it fails, it will notify the administrator.

5.4 DP PTK Trainer

The DP PTK Trainer uses the resource assignments produced by the PTK Resource Allocator for each PTK to train them in the following window: for each PTK_i, it invokes $PTK_i.train\_dp(trainset|^i_{n_i}, \epsilon_i, ...)$, where n_i

Fig.	Predictive Task	Problem	PTKs Used	Model	Model parameters	Baseline
5(a)	Predict ad click on Criteo w/ PTKs from Criteo	binary classif.	39 count featurization PTKs	neural net	VW. One 35 nodes hidden layer with tanh activation. LR: 0.15. BP: 25. Passes: 20. Early Terminate: 1	model w/o PTKs
5(b)	Predict bike station on Citibike w/ PTKs from Taxi	465-class classif.	1 tree PTK	logistic reg.	VW: linear model BP:26. LR: 0.0724702	model w/o PTKs
5(c)	Predict ride duration on Taxi w/ PTKs from Taxi	regression	9 hist. stat., 2 cluster, 2 PCA PTKs	gradient boosting	XGBoost. LR: 0.05. columns sample: 0.5. reg.lambda 3.0. 500 estimators. early stop: 30	model w/o PTKs
5(d)	Predict ride duration on GreenCab w/ PTKs from Taxi	5(c)	5(c)	5(c)	5(c)	5(c)
5(e)	Recommend on Douban Movie w/ PTKs from Movie	regression	1 user embeddings PTK	linear reg.	VW linear model w/ interacted latent features. BP:26 LR:0.015 LRDecay:0.97 PowerT:0 Passes:20	VW matrix factoriz.
5(f)	Recommend on Douban Book w/ PTKs from Movie	regression	User embeddings form Movie	VW mat fact.	VW w/ interacted latent user features & bookID. BP:20 LR:0.015 LRDecay:0.97 PowerT:0 Passes:20	VW matrix factoriz.
Tab. 3	5(c)	5(c)	5(c)	5(c)	5(c)	5(c)

Tab. 2: **Evaluation tasks.** Fig. 5(a) evaluates Scenario 1; Fig. 5(b) evaluates Scenario 2; Fig. 5(c)-5(f) evaluate Scenario 3.

and ϵ_i are the samples and privacy budget allocations produced by the resource allocator for PTK_i. Because each $train\_dp$ function is DP, and the $\mathcal{E}$ global privacy budget is allocated correctly among them, the PTK trainer overall satisfies $\mathcal{E}$-DP.

5.5 Practical Considerations

A few practical consideration follow. First, we left unspecified the size of the profiling set. In general, we envision that set to be the same as the training set used to actually train the PTK. If multiple PTKs need to be profiled concomitantly, one could keep multiples of the training sets in the profile set. Second, we assumed that the PTK developer determines the evaluation function and performance target for each PTK. This can be limiting, because different user tasks may be affected differently by a PTK's error. Sage could support PTK-user-driven performance metrics and targets by forking a new PTK every time a user wishes to customize an existing PTK. The new PTK would then be profiled and allocated resource based on its target.

6 Evaluation

Our questions are: **Q1:** Are PTKs useful as units of data sharing and retention in realistic scenarios such as those in §2? **Q2:** Is Sage's resource allocation procedure effective at producing allocations with minimal training sets?

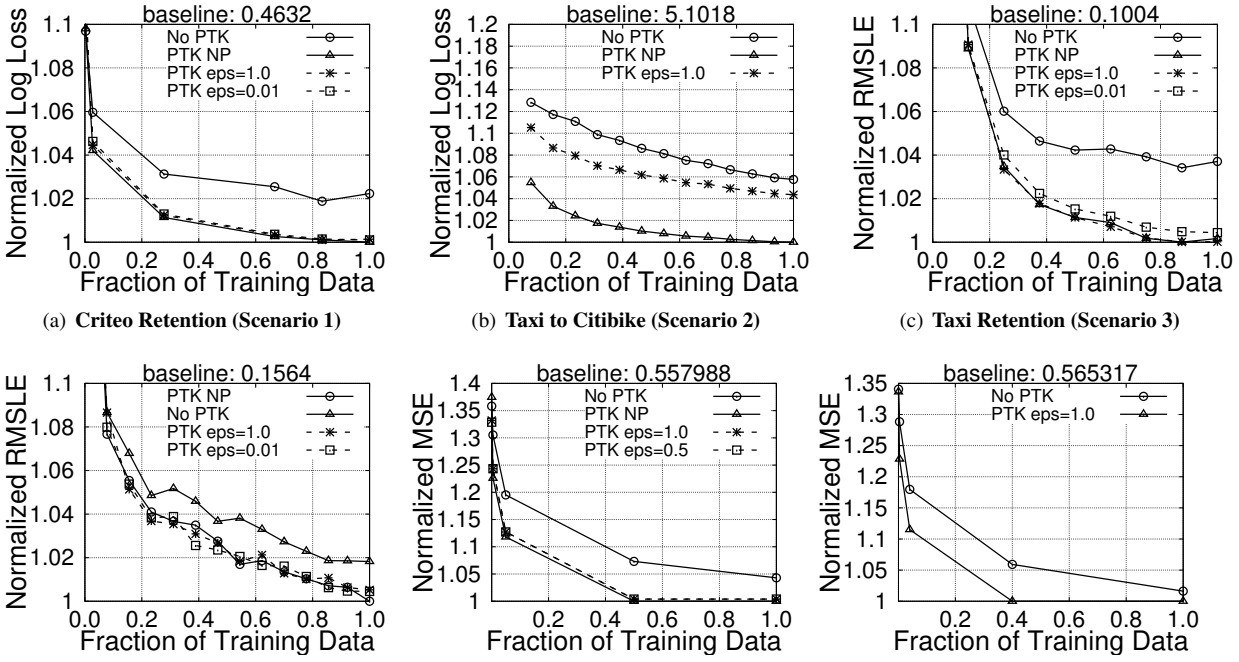
6.1 Methodology

We evaluate PTK usefulness as units of retention and sharing by instantiating the scenarios in §2 on six public datasets (described below). For each scenario, we define predictive tasks, design predictive models that would work well for those tasks, and identify PTKs that might help improve performance for those models. Most datasets were previously included in Kaggle competitions, hence where available, we reuse the tasks and top-performing models from those competitions. Tab. 2 gives details about the prediction tasks we use for each scenario/graph and the models we use. Our choice of scenarios/datasets/tasks/models aims to reflect diversity across tasks (including binary classification, multi-class classification, and regression tasks), models (including linear models, neural networks, boosted trees), and PTKs (all PTKs in Tab. 1 are included in some task), to evaluate wide-applicability of PTKs (requirement **R1** from §3).

Depending on the scenario, we train PTKs on data from either the predictive task's own dataset (retention scenario) or from a separate, related dataset (sharing scenario). We train PTKs on complete datasets and the predictive models on increasing fractions of the datasets. We measure performance of predictive models using standard loss metrics appropriate for each task.

We report most of our results in graphs of the following form: x axis is the fraction of training data given to the predictive model; y axis is the loss metric. We typically plot two baselines and several PTK-based results: *"No PTK"*: first baseline, plotting the predictive model's performance when trained without the PTK features; *"NP PTK"*: second baseline, plotting the model's performance when trained with non-DP PTK features; *"PTK ϵ"*: PTK-based results, plotting the model's performance when trained with ϵ-DP PTK features.

Datasets. • *Criteo [2]:* Ad click/noclick log from users of Criteo ad company: 39 million ad impressions over seven days. Each entry has 26 categorical features, 13 integer features, and a binary label indicating if the ad was clicked. We use the same task as a Kaggle competition on this dataset: predicting whether a user will click on an ad. We adopt the top-performing predictive model: a neural network binary classifier [1]. • *Taxi and Green Cab [8]:* Two separate datasets released by the NYC Taxi Commission from its Yellow and Green cab divisions, resp. The datasets consist of taxi rides in NYC and have 13 features in common, including: start and end location, time of day, and weather information. We use data from the same three months in 2015 across the datasets: 37,105,381 rides in Taxi and 4,850,903 rides in Green Cab. We use the same task as a Kaggle competition organized around the Taxi dataset: predicting the taxi trip duration. We adopt the top-performing model: an XGBoost model [7]. • *Citibike [5]:* Dataset released by the NYC Citibike bike renting system, consisting of bike trips between the company's docking sta-



(a) **Criteo Retention (Scenario 1)** (b) **Taxi to Citibike (Scenario 2)** (c) **Taxi Retention (Scenario 3)**

(d) **Taxi to Green Cab (Scenario 3)** (e) **Douban Movie Retention (Sc. 3)** (f) **Douban Movies to Douban Books (Sc. 3)**

Fig. 5: **PTK Usefulness.** Task-specific loss normalized to the best baseline with non-private PTK on the complete dataset. Baseline value on top.

tions in NYC. We use a three month dataset from 2015 that consists of 3,489,222 rides. The dataset has 13 contextual features, all common with Taxi and Green Cab datasets. • *Douban Movies and Books [6]:* Two datasets with ratings of movies and books, resp., by 36,673 users of the Douban Chinese media company. Consist of 5,257,666, and 2,578,947, resp. ratings over a period of three months. Each rating entry has two features: user ID and movie/book ID. User IDs match across the datasets.

6.2 Scenario 1 (Retention)

We instantiate Scenario 1 on the Criteo dataset. Nine of Criteo's 26 categorical features are high-dimensional (10K or more values). We hypothesize that count featurization, which reduces dimensionality of categorical features, will help improve the rate of learning for this predictive task and make the predictive model converge using less data. This will let the company reduce its retention period without losing accuracy. We therefore register a count featurization PTK and let it train over the first 80% of the dataset. We test on the final 20%. We sum the DP counts across windows for featurization. We train the neural network classifier using the count features in addition to all of the raw features.

Fig. 5(a) shows our results. We find that training the predictive model with the PTK ($\epsilon_{train} = 1.0$) requires just *25% of the dataset* to match the best performance achieved by the non-PTK model trained *on the entire dataset*. Moreover, the difference between using the protected PTK features and using the unprotected count features is negligible ($< 0.14\%$ at $\epsilon = 1.0$). Thus, in this scenario, the PTK is a useful unit for long-term retention and there is no reason for the primary task to retain historical counts in unprotected form.

6.3 Scenario 2 (Sharing)

We use Taxi as the dataset available to the taxi division in our scenario and Citibike as the dataset available to the new bike division. We define PTKs on Taxi (detailed below) and transfer them to the Citibike transfer task. The Citibike task predicts the destination bike station of each ride out of 465 possible stations that the Citibike company has in NYC. It is given all features except end location, which is the predicted label.

The goal is to develop powerful PTK features, learn them over the extensive Taxi dataset, and transfer them to the newly added Citibike task to help bootstrap it with as little data as possible. We observe that the Taxi dataset is amenable to tree modeling, so we define a tree PTK on the dataset that uses the common features (except destination), to predict the closest destination bike station for each taxi ride. We use the tree PTK to generate 465 new features for each Citibike ride that correspond to the predicted probability of each the 465 destination stations. We use the featurized CitiBike data to train a logistic regression for the Citibike task.

Fig. 5(b) shows the normalized loss for the Citibike task, respectively, as the amount of training data supplied to these tasks increases. We find that while the task has limited data (up to 54% of the dataset, or about 49 days), adding the Taxi tree PTK improves by about 2.1% at $\epsilon = 1.0$ ("PTK $\epsilon = 1.0$" vs. "No PTK"). By the end

of the dataset (three months), the difference shrinks to 1.3%. In both cases, the choice of transferring the tree features safely with $\epsilon = 1.0$-DP incurs quite a bit of overhead compared to transferring the unprotected tree features ("NP PTK"). Still, the PTK helps bootstrap a *new, related transfer task* with limited data.

6.4 Scenario 3 (Envisioned Sage Use)

Coming up with good features is hard. Wouldn't it be great if a team could reuse another team's features for their own purposes, without exposing the data? We answer on two datasets: NYC Taxi and Douban.

Scenario on NYC Taxi. Assume a transportation company has two separate teams, Yellow Cab and Green Cab, each with access to two separate datasets: Taxi and Green Cab. They have the same predictive task: predicting the duration of a ride. The Yellow Cab team develops "good" features for its own task and shares them as PTKs through Sage; the Green Cab subscribes to them and evaluates whether they are useful as-is for their task.

To design "good" features for the Taxi task, we get inspiration from the top-performing model of the Kaggle competition organized around the same task on the Taxi dataset. In addition to 13 contextual features, the model uses 9 historical aggregate features related to the average speed of trips, clusters of all pickup locations and drop off locations, and PCA over the pick up and drop off location. We convert these aggregate features into PTKs and train them on the entire Taxi dataset.

We first confirm that these are indeed "good" features for the Taxi task, even as they are made DP. Fig. 5(c) shows the normalized loss for the duration prediction task on increasing fractions of the Taxi dataset. Using the historical features (with DP or without) is useful for this task: almost 3.5% improvement of using the $\epsilon = 1.0$ PTK compared to not using aggregate features at all. Making the historical features DP with the help of PTKs affects performance extremely little, even at $\epsilon = 0.01$. Thus, it is plausible that the Taxi team would be willing to develop and use these PTKs for their own task.

We next share the PTKs over Sage and evaluate whether the Green Cab team benefits from leveraging them in their own, related task. Fig. 5(d) shows the results. Using the Taxi PTKs in the Green Cab task is considerably better compared to not using any aggregate features (1.27% performance improvement for $\epsilon = 1.0$). Training their own aggregate features on Green Cab data results in slight improvement, but there is a convenience argument for leveraging PTKs that already exist.

Scenario on Douban. We ran a similar scenario on the Douban datasets. Imagine a media company has two separate teams, Movies and Books, each with access to the respective datasets. The Movies team leverages the private user embedding PTK with $\epsilon = 1.0$ to create latent features in the movies dataset. The Books team uses those latent features in conjuction with latent features learned locally using VW. Fig. 5(f) shows that movie latent features help improve prediction performance by 5.9% when trained on 40% of the data and by 1.6% when trained on the entire dataset. Private user embeddings can also be used to improve retention and exposure on the same dataset. They are thus useful retention abstractions for the Movie team. Fig. 5(e) shows the results of using the same embeddings as features interacted with user and book IDs in a linear SVM trained with VW on the Books dataset. Featurization with the Movies PTK improve the Books model by 4.9%.

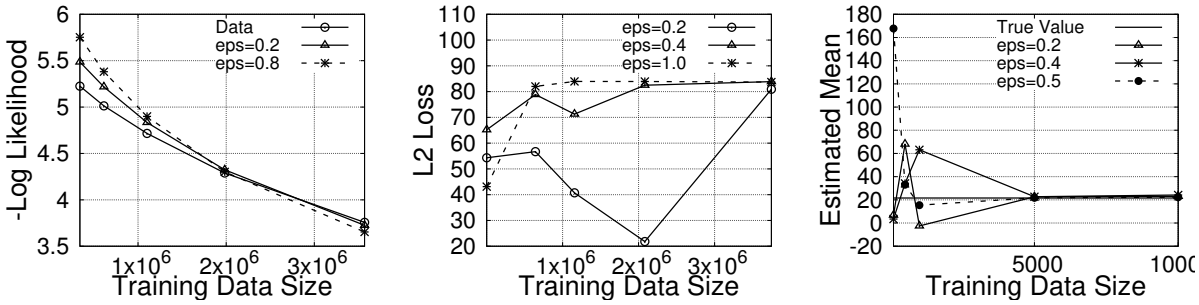
6.5 Resource Allocation Evaluation

We evaluated the PTK resource allocator on the Taxi duration prediction task, which uses a total of 13 PTKs. We produced the PTK profiles with both loose and tight values of τ specific to each PTK. We noticed that the impact of the choice of τ for the features was limited on the the predictive model's performance. This is not surprising, because it is known that features, even if not extremely predictive on their own, can still add significant value to predictive models (ensemble-based models are an example of that effect).

In our experiment, we ran the allocation procedure on profiles with a loose value of τ and $\mathcal{E} = 1$. Using the profiles, the allocation procedure returned an assignment with a single partition ($K = 1$). The total number of samples estimated to be needed was 357,009, or less than one day of data! This is much shorter than we imagined based on our experience with the predictive model, which continues to improve performance even after 40 or so days' worth of training data. This suggests that Sage can maintain an extremely rigorous retention policy.

Tab. 3 shows the ϵ_i, n_i allocations the PTK Resource Allocator assigned to each PTK. Most historical statistics receive $\epsilon = 0.1$ privacy budget. The clustering PTKs are assigned $\epsilon = 0.2$ because the clusters. They are the most data-hungry PTK in this workload. Finally, $\epsilon = 0.5$ of the privacy budget is allocated to one historical statistic: the average speed between locations. Many of these locations are extremely sparse (have very few taxi pickups in them), hence they need more privacy budget to meet reasonable performance. Training the predictive model on the preceding allocation resulted in 0.7% performance penalty over the best performance we got, which used three months of the dataset.

PTK Profiles. We end with some interesting insights we observed from looking at raw (non-DP) PTK performance profiles. Fig. 6 shows the performance/sample complexity tradeoff of three PTKs at various values of the privacy budget: private embeddings, private k-means, and private historical statistics. These three PTKs show 3 distinct behaviors highlighting the need for PTK profiling. The private embeddings profile in



(a) **User Embedding Profile (Douban)** (b) **K-Means Profile (Taxi)** (c) **Historical Statistics Profile (Taxi)**

Fig. 6: **PTK Profiles.**

PTK	τ	n_i	ϵ_i
Hist. Stat.	10	3005	0.01
2 Hist. Stat.	10	1224	0.01
2 Clusters	n/a	357009	0.2
2 PCA	0.1	199	0.01
3 Hist. Stat.	70	16942	0.01
Hist. Stat.	70	41628	0.01
Hist. Stat.	70	15674	0.01
Hist. Stat.	70	88401	0.5
	Total:	357009	

Tab. 3: **PTK Resource Allocation (Taxi).**

Fig. 6(a) shows that performance continuously improves as we add data. Private k-means exhibits a very different behavior (Fig. 6(b)). The L_2 loss of the clusters fluctuates when the clusters are trained with little data or with a low privacy budget. The clusters do converge when trained on a large amount of data and change very little when more training data is added. This indicates that Sage should allocate an adequate amount of data to ensure convergence but no more. The last PTK exhibits yet another behavior. This PTK is estimating the average speed of taxi rides during an hour of the data. The deviation from the true value becomes more accurate as it is computed on more data. The historical statistics maintain their performance once computed on sufficient samples.

7 Related Work

Sage Semantic vs. Standard DP Semantics. Sage's threat model and semantic relate closely to standard models based on DP. Sage's global privacy semantic satisfies $\mathcal{E}$-DP under continual observation [19]. Sage's minimal-exposure semantic relates to pan privacy [18]. Sage's semantic, which permits $\mathcal{T}$-bounded leakage, is weaker than pan privacy, which permits no leakage.

Relationship with DP Literature. The vast majority of DP and DP ML literature has focused on *algorithm* design: most ML and data mining algorithm now have DP versions [37, 9]. We leveraged this fact when designing the PTK abstraction and developing ptklib. A few works have tackled *systems* aspects, such as managing multiple DP queries/models under a global privacy semantic [53, 34, 40]. The approach has thus far been to allocate a *single resource* – the global privacy budget – across the queries/models running on a database. Works here fine-tune the allocation to query sensitivity to noise, assigning more budget to more sensitive queries [53, 34]. Although the idea of partitioning the data has been used in many DP settings (e.g., the sample and aggregate framework [43], private ensembles of teachers [46]), no one has formulated the problem of managing the global privacy semantic for a set of queries/models as *a double-resource allocation problem* that allocates not only *privacy budgets* but also *sample sizes* to queries/models based on their sensitivity. The closest work to doing so is GUPT [40], which fine-tunes a block size in the sample-and-aggregate framework for computing real-valued statistics, but their framework does not extend to arbitrary differentially-private computations.

Sample Complexity Estimation in Statistics. The central problem faced in our PTK profiling is the fact that tight bounds on sample complexity for (private) learning are not generally known *a priori*, as they depend on the data distribution in complex ways [12]. Our approach uses techniques from model selection [36], although they have not been used in this context before.

8 Conclusion

Machine learning systems have very different data access and sharing needs than traditional software. Yet, these systems have inherited protection abstractions from traditional systems (e.g., access control on data files, tables, or streams) that are ineffective at supporting the sharing patterns and force companies to adopt either overly loose or overly tight access policies. We proposed *private transferrable knowledge* (PTK), a new protection abstraction designed to support this new class of applications. PTKs are differentially private feature models that can be used to (1) limit data exposure timeframes within primary tasks that require access to raw data and (2) enable protected data sharing with transfer tasks that do not require raw data access. To manage PTKs, we presented *Sage*, the first DP ML system to maintain a global DP semantic across multiple models trained on the same data streams by allocating not only the privacy budget (a non-scalable resource) but also the number of samples from the data streams (a more scalable resource).

References

[1] https://www.kaggle.com/c/criteo-display-ad-challenge/discussion/10429#54591, 2014.

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

[3] Data lakes and the promise of unsiloed data. http://usblogs.pwc.com/emerging-technology/data-lakes-and-the-promise-of-unsiloed-data/, 2015.

[4] Lessons from integrating machine learning models into data products. https://www.slideshare.net/SharathRao6/lessons-from-integrating-machine-learning-models-into-data-products, 2017.

[5] Citibike system data. https://www.citibikenyc.com/system-data, 2018.

[6] Douban dataset. https://sites.google.com/site/erhengzhong/datasets, 2018.

[7] From eda to the top (lb 0.367). https://www.kaggle.com/gaborfodor/from-eda-to-the-top-lb-0-367, 2018.

[8] Nyc taxi & limousine commission - trip record data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml, 2018.

[9] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[10] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, 2016.

[11] 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 *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 273–282. ACM, 2007.

[12] P. L. Bartlett and S. Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[13] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational Inference: A Review for Statisticians. *ArXiv e-prints*, 2016.

[14] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the sulq framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138. ACM, 2005.

[15] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[16] K. Chaudhuri and D. Hsu. Sample complexity bounds for differentially private learning. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 155–186, 2011.

[17] K. Chaudhuri, D. J. Hsu, and S. Song. The large margin mechanism for differentially private maximization. In *Advances in Neural Information Processing Systems*, pages 1287–1295, 2014.

[18] C. Dwork. Differential privacy in new settings. In *Symposium on Discrete Algorithms (SODA)*, January 2010.

[19] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 715–724. ACM, 2010.

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

[21] C. Dwork, K. Talwar, A. Thakurta, and L. Zhang. Analyze gauss: optimal bounds for privacy-preserving principal component analysis. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 11–20. ACM, 2014.

[22] J. Eng. OPM hack: Government finally starts notifying 21.5 million victims. http://www.nbcnews.com/tech/security/opm-hack-government-finally-starts-notifying-21-5-million-victims-n437126, 2015.

[23] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015.

[24] A. Friedman and A. Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–502. ACM, 2010.

[25] P. Gopalan, J. M. Hofman, and D. M. Blei. Scalable recommendation with hierarchical poisson factorization. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, pages 326–335, Arlington, Virginia, United States, 2015. AUAI Press.

[26] S. Gorman. NSA officers spy on love interests. `http://blogs.wsj.com/washwire/2013/08/23/nsa-officers-sometimes-spy-on-love-interests/`, 2013.

[27] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[28] T. Gryta. T-Mobile customers information compromised by data breach at credit agency. `http://www.wsj.com/articles/experian-data-breach-may-have-compromised-roughly-15-million-consumers-1443732359`, 2015.

[29] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9. ACM, 2014.

[30] N. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.

[31] N. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for sql queries. *Proc. VLDB Endow.*, 11(5):526–539, Jan. 2018.

[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. Laur, H. Lipmaa, and T. Mielikäinen. Cryptographically private support vector machines. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 618–624. ACM, 2006.

[34] M. Lecuyer, R. Spahn, R. Geambasu, T.-K. Huang, and S. Sen. Pyramid: Enhancing selectivity in big data protection with count featurization. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 78–95. IEEE, 2017.

[35] L. E. Li, E. Chen, J. Hermann, P. Zhang, and L. Wang. Scaling machine learning as a service. In *Proceedings of The 3rd International Conference on Predictive Applications and APIs*, volume 67 of *Proceedings of Machine Learning Research*, pages 14–29, Microsoft NERD, Boston, USA, 11–12 Oct 2017. PMLR.

[36] P. Massart. *Concentration inequalities and model selection*, volume 6. Springer, 2007.

[37] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the Netflix prize contenders. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636. ACM, 2009.

[38] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 94–103. IEEE, 2007.

[39] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 19–30, New York, NY, USA, 2009. ACM.

[40] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. GUPT: Privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 349–360. ACM, 2012.

[41] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. Graphsc: Parallel secure computation made easy. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 377–394. IEEE, 2015.

[42] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 334–348. IEEE, 2013.

[43] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual*

ACM symposium on Theory of computing, pages 75–84. ACM, 2007.

[44] C. Ornstein. Celebrities medical records tempt hospital workers to snoop. https://www.propublica.org/article/clooney-to-kardashian-celebrities-medical-records-hospital-workers-snoop, 2015.

[45] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[46] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*, 2018.

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

[48] D. Shiebler and A. Tayal. Making machine learning easy with embeddings. 2015.

[49] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 3–18. IEEE, 2017.

[50] A. Shrivastava, A. C. Konig, and M. Bilenko. Time adaptive sketches (ada-sketches) for summarizing data streams. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1417–1432. ACM, 2016.

[51] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin. Differentially private k-means clustering. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 26–37. ACM, 2016.

[52] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.

[53] X. Xiao, G. Bender, M. Hay, and J. Gehrke. iReduct: Differential privacy with reduced relative errors. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 229–240. ACM, 2011.

[54] A. Zheng and A. Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. 2018. OCLC: 1029545849.