Protection and the Control of Information Sharing in Machine Learning Systems

Anonymous Submission

Abstract

Machine-learning (ML) ecosystems that support today's organizations lack appropriate abstractions to protect, control sharing of, and retain sensitive information. 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 a unit of protected data sharing and retention. We build *Sage*, a PTK store that creates, maintains, and shares multiple PTKs on behalf of a wide range of ML workloads. Sage's uniqueness lies in its minimal-exposure design, which conflicts with the global privacy semantic it seeks to enforce for PTKs. We address this tension with new statistical and systems methods.

1 Introduction

Data-rich, machine learning (ML) ecosystems arising in today's companies, governments, and organizations forgo important principles for rigorous data protection. Consider the *least privilege principle* introduced by Saltzer in the 1974 Multics paper [**?**]: *"[e]very program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job."* This principle has influenced the design not only of the Multics protection system, but also of UNIX and, through that, of many modern operating systems, including Linux, OSX, and Windows. In today's data-driven world, this principle appears all but forgotten. Take, for instance, the "data lake" architecture that is emerging in many ML ecosystems [**?**, **?**]: user data from a company's multiple products is pooled and integrated into a single, giant repository, which archives it for indefinite time periods and makes it accessible to every data engineer and service within the company who might have a use for it. While this architecture fosters innovation, it also flaunts the least privilege principle and exposes data stores to "unintentional, unwanted, or improper uses of privilege" by employees or hackers [**?**].

Many companies have adopted casual protection principles due to the lack of appropriate data protection, sharing, and retention abstractions for ML workloads. 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 necessary to implement certain functionality. 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 needs access to the user account database; the social networking service needs access to the users' friends lists and posts but not to the account database or credit card information; and an in-app product purchase service needs access to credit card

information but not to the social graph or account information. To meet the principle of least privilege, the company could store these different data types in different database tables and enable access to the various services and their engineering teams strictly on a needs basis.

In contrast, emerging ML-driven applications blur the issue of whether a particular type of data is "necessary" for some functionality. For example, data collected to improve a post recommendation service – including the posts previously read or liked by each user – appear relevant not only for that service but also for a friend recommendation service, an ad targeting service, and potentially even for a bot detection service. Similarly, data collected for ad targeting – such as ad clicks or visited Web pages – may be relevant both for the post and friend recommendation services and for the bot detection service. In some cases, the data may ultimately prove useless; however, knowing whether it is useful requires its use – and therefore access to its files or tables – for experimentation and possibly evaluation in production.

These blurry boundaries between what data is and is not necessary for various services and teams within a data-driven company create tension between an interest in enabling wider access to data and concerns about privacy, ethical, and unnecessary exposure of information. Different companies resolve the tension differently, but each solution imposes a cost. Some companies prefer to silo sensitive data streams to the detriment of functionality. Others embrace wider-access, data-lake architectures to the detriment of data protection. And yet others choose to share some data streams while siloing others. Regardless, it remains the benefit/cost tradeoff varies for each scenario as do the opportunity costs for not leveraging data when it would in fact be useful to do so.

We believe that this tension can be resolved by the development and use of new data protection abstractions that are better suited to emerging ML workloads than traditional files, tables, or data streams. We observe that in many data sharing scenarios, it is not the raw data whose access is actually needed (or even desirable), but "knowledge" drawn from that data, such as historical statistics, well-engineered features, or in general, *models* trained on the data. For example, the friend recommendation service may benefit more from incorporating user embeddings already trained and curated over the post service's data, rather than the raw posting activity of the users. These embeddings, which map each user to a low-dimensional space that captures the similarity among users in terms of the posts they read, write, and like, are usually easier to incorporate in new machine learning tasks compared to the raw data, which is highly dimensional and messy. User embeddings are often shared across teams at big companies, including Twitter [**?**].

We propose the new *private transferrable knowledge* (PTK) abstraction, 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. PTKs are meant to be used as a protected unit both for long-term retention by the teams that need raw access to some stream of personal data and for sharing of information from that stream with teams that do not require raw access. 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. Two considerations motivate this focus. First, engineering good features is a very challenging and time consuming, but extremely important, part of any ML ecosystem. Appropriate features make model training and optimization easier and less data intensive, because the features already capture important historical characteristics. Second, there is increasing evidence that sharing good features across teams is common practice in today's data-driven companies. Uber and Instacart have developed *Feature Stores* where teams share tens of thousands of curated features [**?**, **?**, **?**]. For these reasons, we believe that elevating feature models to the rank of protection abstraction in ML systems is a good step toward reenacting rigorous protection principles in these systems.

To implement PTKs, we developed the *Sage PTK Store*, a minimal-exposure system that manages, trains, and makes available multiple PTKs on an organization's data streams. Sage reinvigorates the least privilege principle by letting the company share PTKs widely across teams to fulfill feature sharing needs and share raw data only with services that truly need it.

Sage's design resembles existing feature stores, with some notable differences. As is the case for existing feature stores, the developers in charge of a data stream define feature-learning procedures and register them with Sage for continuous computation. Sage ingests data from these streams to train these feature models and make them available to others, who incorporate them into their predictive models. Unlike existing feature stores, Sage: (1) accepts only feature learning procedures that preserve DP guarantees, and (2) strives to minimize the exposure of the raw data it ingests to avoid violating the least privilege principle or introducing new risks for the organization. For (1), we leverage directly the enormous body of existing work on DP learning algorithms [**?**] to develop an initial PTK library, which we envision as extensible over time to cover most popular feature learning techniques. (2) raises substantial challenges not currently addressed in DP ML literature.

Specifically, minimizing the amount of data Sage must retain, and therefore expose, to train multiple PTKs with acceptable performance is at odds with enforcing a tight global privacy semantic across PTKs. This tension increases as the number of PTKs grows. To mitigate this conflict, we developed two new mechanisms: (1) the *first generic method to estimate rigorous sample complexity bounds for DP models* that fit the general empirical risk minimization framework; (2) the *first dual-resource allocation method for DP systems* that assigns not only privacy budgets 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 performance. These methods make Sage the *first minimal-exposure DP ML system* and a model for least-privilege design.

Using three corporate-inspired scenarios instantiated on six publicly accessible datasets, we demonstrate that PTKs are *useful units for data protection*, enabling many sharing scenarios without needing to share raw data, and allowing ML tasks to gain the benefits of historical features without needing to retain data long term. We also demonstrate that Sage can train effective PTKs while enforcing strict global privacy semantics. In all scenarios, using DP features instead of unprotected ones (akin to features in a vanilla feature store) affects the performance of predictive models by

at most [XX%]. Thus, PTKs are not only a useful abstraction but also a practical one that can be implemented in a data-rigorous way. xxx

2 Use Cases

Scenario 1: A media company collects streams of user activity from three services: a movie streaming service, a music store, and a digital bookstore. User rating, comment, and viewing streams are currently siloed per service, where they are used for recommendations. The company hypothesizes that user preferences are consistent across the three media types. They would like to compute user embeddings across the three streams and incorporate them into reach service-specific recommender. They would like to do so *without* exposing the raw data streams to all services because they worry about potential future break-ins to these services.

Scenario 2: 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 help bootstrap forecasting bike demand in different areas of the city, because they expect that the taxi and bike share data will be used for similar trips. To limit the data's exposure, they wish to share the taxi information privately. If they are successful, the company plans to use the same sharing method in other company expansions, such as to bootstrap a new taxi fleet in the city's boroughs.

Scenario 3: 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 wish to minimize the amount of ad click data they retain at any time. 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 reducing the amount of raw data used to train their predictive models at any time.

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

Beyond this goal, 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 workloads.

R2: *Limited impact on accuracy and performance.* 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 performance, we aim for substantial improvement to justify computational overhead. In all cases, performance overhead for predictive models should be small.

R3: *Minimal-exposure system.* The principle of least privilege should apply to the system that implements this abstraction (Sage). Sage should in fact be a model of rigorous data management and minimize the amount of data it exposes through its internal data 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 can 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: [**?**], [**?**].

Our abstraction is designed to enable companies to 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 the compromisable services running them. Sharing of 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 the parameters of ML models, can reveal a significant amount of information about individual examples in their training sets [**?**]. 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.

One alternative mechanism would be to apply *cryptographic methods*, such as homomorphic encryption (HE) or secure multi-party computation (MPC), to traditional protection units, such as files or streams. Consider an encrypted data lake architecture, where raw data is shared widely but in homomorphic-encrypted form, and teams and services design and train models "blindly" using homomorphic training algorithms [**?**, **?**]. Alternatively, consider a siloed architecture, where raw data is accessible only to the team that collects it, and predictive models are trained (again, blindly) on one or more datasets using MPC. Putting aside the significant performance concerns of running even state-of-the-art HE and 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 [**?**, **?**]. Fixing this requires either an additional MPC protocol with the end users [**?**], which adds to the performance concern, or a combination of HE/MPC with DP. Thus, we view DP as a rather crucial protection mechanism in ML systems.

As an alternative unit of protection, we could consider applying DP at the level of *SQL queries*. For example, consider a company that needs all access to raw data to be done through a DP SQL interface, such as PINQ [**?**] or FLEX [**?**]. This approach works well for workloads easily written on SQL, such as simple aggregates or learning algorithms that can run efficiently on statistical queries [**?**]. However, implementing broad classes of learning algorithms, including those based on stochastic gradient descent, on SQL would be challenging and inefficient. Generally, we believe that for ML workloads, the SQL interface is too low-level an abstraction at which to enforce protection. We therefore elevate the level of abstraction to *machine learning models*, and specifically, to *feature models*.

3.3 Differential Privacy Explained

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.

DP is characterized by its three important properties. (1) *Post-processing resilience*: any computation applied on the output of an ϵ-DP algorithm remains ϵ-DP [**?**]. (2) *Composability 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 [1]. (3) *Resistance to auxiliary information*: regardless of external knowledge, an adversary with access to the outputs of a DP algorithm draws the same conclusions about the presence/absence of a record in the input dataset [**?**]. A corollary of (3) is that if two DP algorithms, ϵ_1-DP A_1 and ϵ_2-DP A_2, use as input two non-overlapping datasets (e.g., two subsets of iid samples from a longer set), the combined privacy semantic is $max(\epsilon_1, \epsilon_2)$-DP.

3.4 Scope

The use cases we aim to support with our abstraction correspond to well-understood and common multitask and transfer learning scenarios. **[Daniel: can you give a bit of background on multi-task and transfer learning and when they are useful?]** xxx

4 The 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 []. This encourages us to believe 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 [**?**] and increasing experimental evidence suggests that DP is amenable to ML [**?**, **?**]. This encourages us to believe that DP can be applied to feature models with limited accuracy overheads (requirement **R2**). Finally, §5 discusses how to implement and manage PTKs effectively while minimizing exposure of the data used to train them (requirement **R3**). PTKs are (1) *units of data sharing*, improving performance in transfer tasks that leverage their features without requiring access to raw data, and (2) *units for long-term retention*, improving performance in primary tasks by letting them use historical information without retaining long-term access to those streams.

4.1 API

Fig. 1 shows the PTK API, which is used by the *PTK developer* and one or more *PTK users*.

PTK Developer. The developer first defines a procedure to learn useful features from an accessible data stream and then turns the feature learning procedure into a PTK by implementing the PTK developer API (described below). The

```
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.**

developer then identifies a reasonable performance target τ for the PTK, which Sage attempts to honor when it assigns privacy budgets and data samples to train the PTKs on overlapping streams. The final step is registering the PTK with Sage and giving Sage access to the source raw data stream on which it will train. 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 the 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 Developer API. We designed the PTK API with simplicity in mind. It consists of two main 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 input data stream (§5). We support continuous PTK training across windows by supplying `ptk.train_dp` with the previous state of the model trained from the previous window. `ptk.eval` is a

	PTK	Type	ptk_train_dp	ptk_eval	Used in
1	**User embeddings**	S	DP Poisson factorization []	loss on holdout set	Scenario 1
2	**Tree featurization**	S	XXX [**?**]	loss on a holdout set	Scenario 2
3	**Count featurization**	S	DP contingency tables [**?**]	error from true value	Scenario 3
4	**Covariance matrix**	U	XXX []	error from true value	Scenario 2
5	**Historical statistics**	U	keyed aggregates w/ Laplace noise []	error from true value	Scenario 2
6	**Clustering**	U	DP Lloyd algorithm []	XX	Scenario 2

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

PTK evaluation procedure that takes as input a testing set and returns some PTK-specific evaluation metric, such as a loss metric for a supervised PTK. The `eval` function need not satisfy DP. $\epsilon_{profile}$.

4.2 Example PTKs

We implemented *ptklib*, a library of PTKs for popular feature learning algorithms. We use a recent feature engineering textbook [**?**] 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`; sensible performance metrics for `ptk.eval`. We exemplify our implementation of three PTKs particularly relevant for our evaluation.

User Embeddings PTK. [TODO(mathias)] Overview of what an embedding is. In an instantiation of Scenario 1, §6.1.1 uses this PTK to improve performance of several tasks in a multitask learning setting. xxx

PTK API: `ptk.train_dp` `ptk.eval`

Tree Featurization PTK. Tree featurization is a technique previously used at Facebook to improve performance of ad click prediction and has an implementation in scikit-learn. It is used to develop informative nonlinear features by constructing a forest of decision trees. Decision trees are functions described by nested if-then-else statements, where the predicates are typically of the form $x_i < \theta$ for continuous features x_i and $x_i = k$ for categorical features x_i, and the final return value is one of the possible label values. Each root-to-leaf path in a decision tree corresponds to a conjunctive (and hence nonlinear) predicate of the input. If the source dataset is amenable to decision trees, then root-to-leaf paths will 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. In an instantiation of Scenario 2, §6.1.2 uses this PTK to bootstrap a new target 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 the decision trees private [**?**]: on every node, we: (1) choose a random subset of the features to split on; (2) compute the predictive utility of all splits over these features; and (3) choose a noisy version of the best splits using the least margin mechanism [**?**], a variant of the more popular exponential mechanism []. The ϵ_{train} privacy budget is split 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 on the test set.

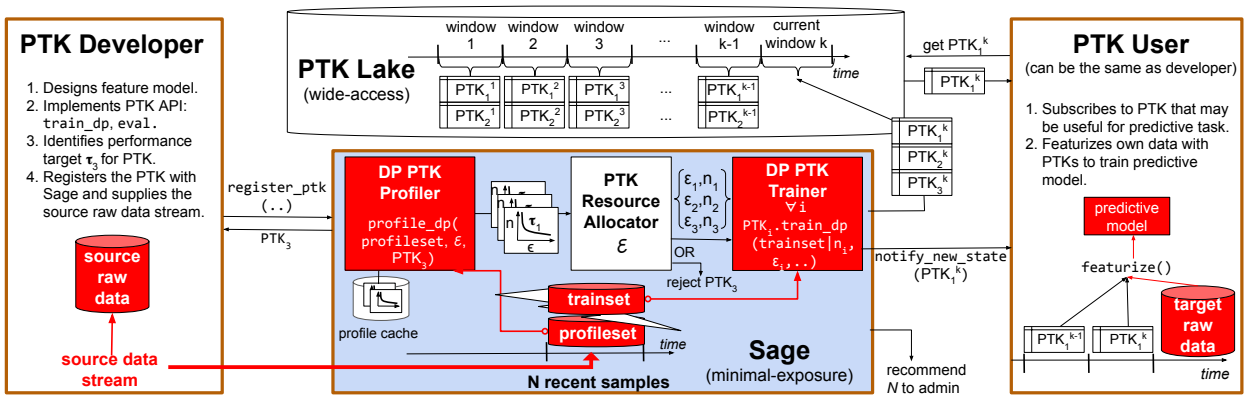


Fig. 2: **Sage Architecture.** PTK_i: PTK object; PTK_i^j: the state of PTK_i at time window j; $trainset|_{n_i}$: n_i iid samples from trainset.

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 [**?**, **?**]. 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 3, §6.1.3 shows how to use this PTK to reduce data retention in primary tasks.

PTK API: `ptk.train_dp` computes DP contingency tables of each feature separately with each label in a particular data stream. It makes the tables DP by initializing their cells with random draws from a Laplace distribution [**?**], scaled by ϵ_{train}. `ptk.eval` returns XXX.

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 Semantics. To protect user data against adversaries listed in §3.1, Sage enforces two protection semantics. First, because PTKs are widely accessible in the data lake, they can be monitored by abusive employees or hackers with access to the PTK 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. 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 $\mathcal{E} = 1$).

Second, because the Sage service ingests multiple data streams, it becomes a point of vulnerability for the company. 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. Unfortunately, enforcing these two semantics while preserving acceptable PTK performance is challenging, particularly as the number of PTKs grows. To satisfy an $\mathcal{E}$-DP global privacy semantic, Sage has two options. One option is to split the global privacy parameter $\mathcal{E}$ (say) equally across the p PTKs, assigning PTK_i a privacy budget of $\mathcal{E}/p$ and the full training window of recent data. This means calling PTK_i`.train_dp(`$trainset|_{\mathcal{N}}$`, `$\mathcal{E}/p$`, ...)` for each $i = 1..p$. This works if the PTKs can tolerate small privacy budgets, which is not always true. For example, while our count featurization PTK trains well even for privacy budgets ≥ 0.001, our tree and user embedding PTKs become worthless at privacy budgets < 1. Regardless, this is not a scalable option for large p.

The other option is to assign all PTKs full privacy budget $\mathcal{E}$ and split the training window (say) equally. This means calling PTK_i`.train_dp(`$trainset|\mathcal{N}/p$`, `$\mathcal{E}$`, ...)` for each $i = 1..p$, where $trainset|_{\mathcal{N}/p}$ takes $\mathcal{N}/p$ iid samples from the available training set. This lets PTKs achieve their best performance under the given $\mathcal{E}$ global privacy semantic, but will pressure Sage to increase the size of its training window to train acceptable PTKs, thereby weakening its minimal-exposure semantic.

Approach. Our approach is to treat the challenge as a *resource allocation problem*, a common problem in systems, and to leverage the canonical systems approach to address it. The resources here are the privacy budget ($\mathcal{E}$) and the size of the Sage training window ($\mathcal{N}$). The former is a fixed, unscalable resource (cannot be increased as the workload increases); the latter is more scalable, although for minimal-exposure semantics, we wish to minimize $\mathcal{N}$. We need to allocate these resources across p PTKs by assigning each PTK_i privacy budget ϵ_i and a training set consisting of n_i iid samples from the available window of recent data, and invoking PTK_i`.train_dp(`$trainset|_{n_i}$`, `ϵ_i`, ...)` to train it. Our approach, taken from systems, first *profiles* 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; it then finds an allocation $(\epsilon_i, n_i)\forall i$ that preserves the

global privacy guarantee ($\mathcal{E}$) and minimizes the total amount of data needed for training ($\mathcal{N}$) while meeting developer-specified performance targets for the PTKs.

Fig. 2 reflects Sage's key architectural components that implement this procedure: *PTK Profiler*, *PTK Resource Allocator*, and *PTK Trainer*. At every window k, 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 PTK Profiler

The PTK Profiler is responsible for profiling one PTK (or a small number of PTKs), on a heldout dataset (profileset), in isolation from other PTKs. Its goal is 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. In theory, a PTK developer can provide a profiling procedure for her PTK. In practice, profiling DP models is challenging and not something that a developer would be able to find in existing literature. We hence develop the *first generic method for rigorously estimating the sample complexity of a DP model*.

Problem Formulation. For concreteness, we focus on PTKs based on finding a predictor f that approximately minimizes a *risk* $\mathcal{R}(f) = \mathbb{E}[\ell(f, Z)]$, i.e., the expected loss of f on a random example $Z \sim P$ drawn from an unknown distribution P over $\mathcal{Z}$. Here, $\ell(f, z)$ denotes the loss of f on the example $z \in \mathcal{Z}$, which we assume takes values in $[0, 1]$. When $\hat{f} \in \mathcal{F}$ is chosen from a class of functions $\mathcal{F}$ so as to minimize the *empirical risk* $\widehat{\mathcal{R}}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\hat{f}, z_i)$ on a random iid sample $(z_i)_{i=1}^n$ from P, it is well-known that the empirical risk of $\hat{f}$ under-estimates the actual risk of $\hat{f}$ by an amount $\varepsilon_{\mathcal{F},P,n}$ that typically (i.e., with high probability, over the choice of random sample) depends on the sample size n, some measure of the "capacity" of $\mathcal{F}$ (e.g., Rademacher complexity), and the interaction of $\mathcal{F}$ with the data distribution P. For the function classes $\mathcal{F}$ we are interested in, it is known that the functional dependence of $\varepsilon_{\mathcal{F},P,n}$ on n is, up to logarithmic 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 $\varepsilon_{\mathcal{F},P,n} \leq \tau$. However, the dependence of $\varepsilon_{\mathcal{F},P,n}$ on $\mathcal{F}$ and P is much more nuanced, and known *a priori* upper-bounds on $\varepsilon_{\mathcal{F},P,n}$ are notoriously loose due to analytic limitations. So, it is not generally possible to sharply "invert" $\varepsilon_{\mathcal{F},P,n}$ to derive a formula for $n_{\mathcal{F},P,\tau}$.

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. 3: **Dual-Resource Allocation Integer Linear Program.**

Our goal is to use a stream of iid data from P to approximate the minimum sample size $n_{\mathcal{F},P,\tau}$ such that $\varepsilon_{\mathcal{F},P,n} \leq \tau$. Our current methodology is limited in that we achieve this goal only when $\mathcal{R}^* := \min_{f\in\mathcal{F}} \mathcal{R}(f) \leq \tau$. In this case, we are able to find $\hat{f} \in \mathcal{F}$ on the basis of a random sample of size approximately $n_{\mathcal{F},P,\tau}$ such that $\mathcal{R}(\hat{f}) \leq \mathcal{R}^* + \tau \leq 2\tau$.

Because of the limitation described above, 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 infinitesimally small $\delta > 0$ without an exceedingly large sample size. We consider a relaxed goal of correctly declaring that $\mathcal{R}^* > \tau$ only when $\mathcal{R}^* > 2\tau$; this is possible with a sample size around $n_{\mathcal{F},P,\tau}$. If $\tau < \mathcal{R}^* \leq 2\tau$, we are fine with either finding $\hat{f} \in \mathcal{F}$ with $\mathcal{R}(\hat{f}) \leq 2\tau$ (and hence approximately determining $n_{\mathcal{F},P,\tau}$) or correctly declaring that $\mathcal{R}^* > \tau$.

Profiling Algorithm.

DP Consideration. Because PTK profiles are retained for extended periods so they can be reused across multiple training cycles, they need to satisfy $\epsilon_{profile}$-DP. If Sage is configured to profile one PTK per training cycle, then it sets $\epsilon_{profile} = \mathcal{E}$; if it profiles multiple PTKs per training cycle, then it splits the privacy budget equally. For our generic profiling procedure, we XXX add Laplacian noise XXX.

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. In DP literature, allocating privacy budgets to queries to preserve a global privacy semantic is a known problem with numerous methods for fine-tuning the allocation of this single resource based on query size (reviewed in §8). Fine-tuning allocation of *both* privacy budgets and training samples based on query needs has never been done to our knowledge. We hence introduce the *first dual-resource allocation method for DP systems*.

Problem Formulation. Consider we could split the training set into K iid partitions (K given for now). To satisfy the preceding requirement (1), our goal should be to pack the p PTKs into these partitions, and assign each PTK a privacy budget ϵ_i, such that within each partition, the sum of the privacy budgets assigned to the PTKs in that partition is $\leq \mathcal{E}$. If we denote $x_{ik} = \mathbb{1}\{$assign PTK_i to partition k$\}$, then the preceding condition can be written

as: $\forall k. \sum_i x_{ik}\epsilon_i \leq \mathcal{E}$. To satisfy the preceding requirement (2), our goal should be to minimize the sum of the samples in each partition. The PTK profiles give us, for each PTK_i, a mapping between privacy budget ϵ_i and the number of samples n_i needed to achieve its designated performance target. We can therefore write requirement (2) as a minimization problem: $minimize(\sum_k \max_i x_{ik}n_i)$. This problem is NP-hard. However it can be expressed as a mixed-integer linear program (MILP), for which good solvers exist that find approximate solutions for many problems.

MILP Program. The MILP program finds assignments of the ϵ_i, x_{ik} variables that minimize the total sample size (2) subject to the global privacy constraint (1). Unfortunately, as formulated above, the privacy constraint has quadratic dependencies between its variables, which challenges even the best solvers at scale. To remove the quadratic dependency, we discretize the privacy budgets that can be assigned to each PTK into a few possible values common across PTKs (denoted ϵ_j) and define a single set of assignment variables, $x_{ijk} = \mathbb{1}\{\text{assign } PTK_i \text{ to partition k with } \epsilon_j\text{-DP}\}$. The values ϵ_j can come directly from those that were used for profiling, or they can be supplemented with additional values extrapolated from the profiles to give the solver extra degrees of freedom for challenging assignment cases.

Fig. 3 shows our formulation in Sage. The objective and privacy constraint are now linear. The formulation has an additional assignment constraint, which requires that all PTKs be assigned to exactly one partition and privacy budget value, to avoid trivial, unusable solutions that assign all $x_{ijk} = 0$. The formulation remains NP-hard, but in our experience with the Gurobi MILP optimizer [], running the allocation procedure on up to 50 PTKs has been very positive: Gurobi finds the optimal solution in XX minutes (§6.3). We further invoke it multiple times various numbers of partitions to choose the best allocation. For larger workloads, we envision tapping into the traditional resource allocation literature, where the high-level goal is similar – to pack workloads onto machines such that they do not exceed their capacity while minimizing the number of machines being used – and find techniques to make our solution more scalable []. This work lays the foundation for such improvements by introducing the first formulation of the problem of managing DP models in a streaming setting as a *dual-resource allocation problem* that includes a scalable resource – the data – whereas all of the DP literature has viewed the problem as a *single-resource allocation* of the privacy budget, a fundamentally non-scalable resource.

DP Consideration. Because the PTK Resource Allocator operates only on DP profiles, it need not be made DP. This is beneficial because it enables it to run for extensive time without risking exposure of raw data past Sage's $\mathcal{T}$ retention period.

5.4 PTK Trainer

The PTK Trainer uses the resource assignments produced by the PTK Resource Allocator for each PTK to train them in the following window. Because the PTK Profiler produces an already trained PTK for a recently profiled PTK (the witness, trained on the profile set), we generally do not include that PTK in the current training schedule. Instead, we use the previous resource assignment for the current training session and will use the assignment computed on the

current window for future training sessions. This has two advantages. First, it puts the (expensive) resource allocation task out of the critical path of PTK training. Second, it gives the system the ability to adapt to the new data needs of the new allocation. In addition to producing an assignment for each PTK, the PTK Resource Allocator also produces an assessment of the sample size, $\mathcal{N}$, needed to apply the new assignment. That information is passed to an administrator, who can decide whether he is willing to extend the next training window accordingly. Because changing the window size impacts Sage's minimal exposure semantic, we make this an administrative decision.

5.5 Extensions

The Sage design thus far has several limitations, which we could address in future iterations. First, we assumed thus far that all PTKs will be trained once per training window. But different PTKs may have different freshness requirements. Sage can be extended to allow training a PTK multiple times per training window by "replicating" that PTK multiple times in the PTK allocation procedure, and incorporating their deadlines in its formulation. Second, 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. Third, 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 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 targets.

6 Evaluation

We seek to answer three questions in our evaluation:

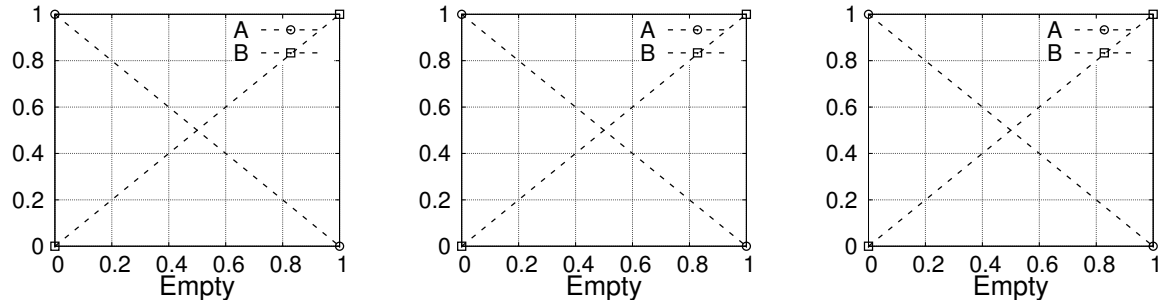
Q1: Are PTKs useful as units of data sharing and retention in realistic scenarios?

Q2: Is Sage's resource allocation procedure effective with many PTKs?

Q3: What are the performance overheads of PTKs and the Sage resource allocation procedure?

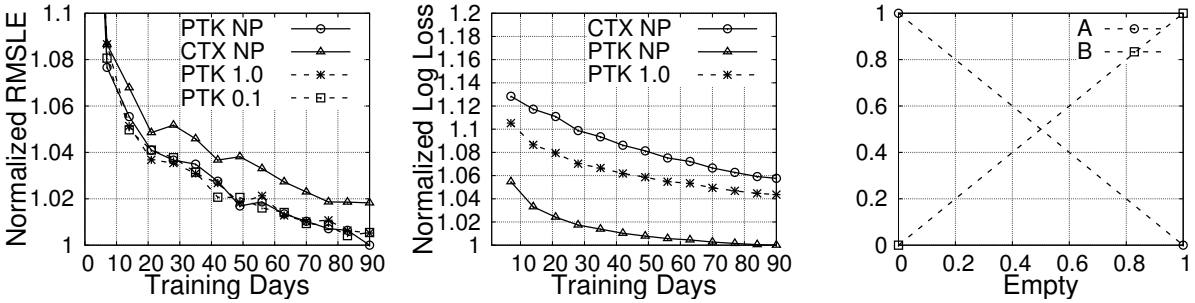
For (Q1), we instantiate the three corporate-inspired scenarios from §2 on public datasets. For (Q2), we use some of these scenarios to evaluate resource allocation under contention. For (Q3), we microbenchmark performance of predictive models (training and prediction) and of our allocation procedure.

We describe specific methodology in each section, however overall, our approach is as follows. First, we use standard performance and accuracy metrics that are appropriate for the various tasks we define. Accuracy metrics include, depending on the task: XXX (explain briefly) and RMSLE (explain briefly). Performance metrics include latency of a given operation. Second, we make explicit efforts to use for each case the strongest baselines we could find. For example, in most PTK usefulness evaluation (Q1), we leverage tasks from previous Kaggle competitions and use as baselines the best-performing models from those competitions. Third, across the use cases, we use varied types of predictive tasks, including regression and binary and multi-class classification.



(a) **Movie Rec. Task** (b) **Music Rec. Task** (c) **Book Rec. Task**

Fig. 4: **Scenario 1 on Douban datasets.**



(a) **Taxi to Green Cab** (b) **Taxi to Citibike** (c) **XXX One more on Taxi?**

Fig. 5: **Scenario 2 on Taxi, Citibike, and Green Cab datasets.**

6.1 Use Case Evaluation (Q1)

We instantiate the three scenarios in §2 on publicly available datasets and evaluate the usefulness of PTKs in each case.

6.1.1 Scenario 1

[TODO(mathias): Please put some notes in this section. I can write the proper text, but I need info. Try to give me all the info I include for the other scenarios.] xxx

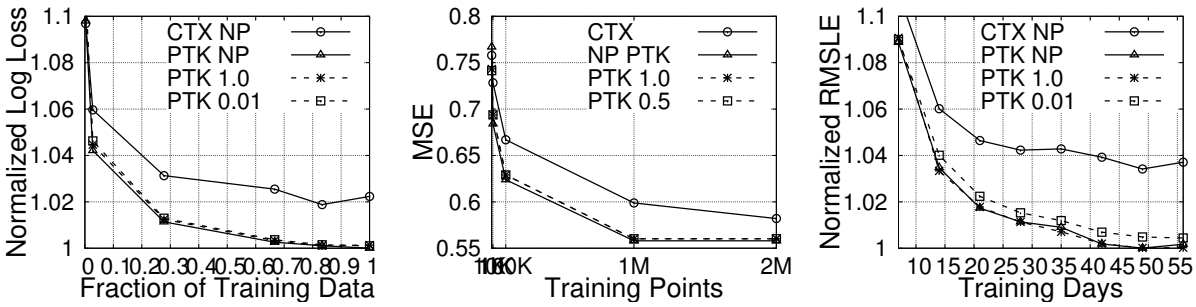
Datasets. We instantiate Scenario 1 on three datasets containing ratings for movies, books, and songs by 36,673 users of the Douban Chinese media company. The datasets consist of XXX, XXX, and XXX ratings for XXX movies, XXX songs, and XXX books, respectively. Each rating entry has two features: user ID and movie/book/song ID. User IDs match across the three datasets. XXX Kaggle competition and model?

Method.

Results. Fig. 4 shows ... Thus, in this multi-task learning scenario, PTKs are a useful unit for sharing and obviate the need to share – and therefore expose – the raw data with the target tasks.

6.1.2 Scenario 2

Datasets. We instantiate Scenario 2 on three public datasets released by two public transportation companies in New York City (NYC): Taxi and Green Cab, two datasets released by the NYC Taxi and Limousine Commission from its Yellow and Green cab divisions, resp.; and Citibike, released by the City-run Citibike bike sharing company. We



(a) **Criteo** (b) **Douban** (c) **Taxi**

Fig. 6: **Scenario 3 on Criteo, Douban, and Taxi datasets.**

use Taxi as the dataset available to the original taxi division in our scenario, Citibike as the dataset available to the new bike division, and Green Cab as the dataset available to the borough-expansion division. The datasets consist of taxi/bike rides in NYC and have 13 features in common, including: start and end location, time of day, day of week, and weather information that day. We use data from the same three months in 2015 across the datasets: 37,105,381 rides in Taxi, 3,362,370 rides in Citibike, and XXX rides in Green Cab. We define PTKs on Taxi (detailed below) and transfer them to Citibike and Green Cab target tasks. 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 Green Cab task predicts how long a ride will take given all features.

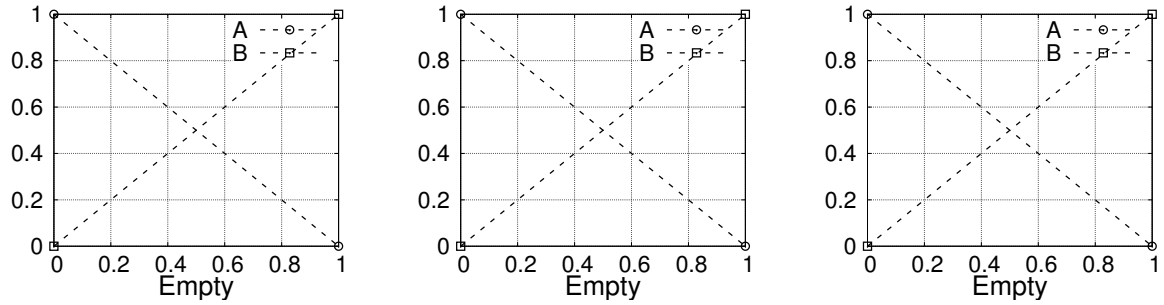
Method. For the Taxi $\rightarrow$ Citibike transfer scenario, we observe that the Taxi dataset is amenable to tree modeling, so we define a tree PTK on the dataset that uses the common features (except destination), to predict the closest destination bike station for each taxi ride. We use the tree 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 Taxi $\rightarrow$ Green Cab transfer scenario, we XXX.

Results. Fig. **??** shows ... Thus, in this transfer learning scenario, PTKs are a useful unit for sharing and obviate the need to share – and therefore expose – the raw data with the target tasks.

6.1.3 Scenario 3

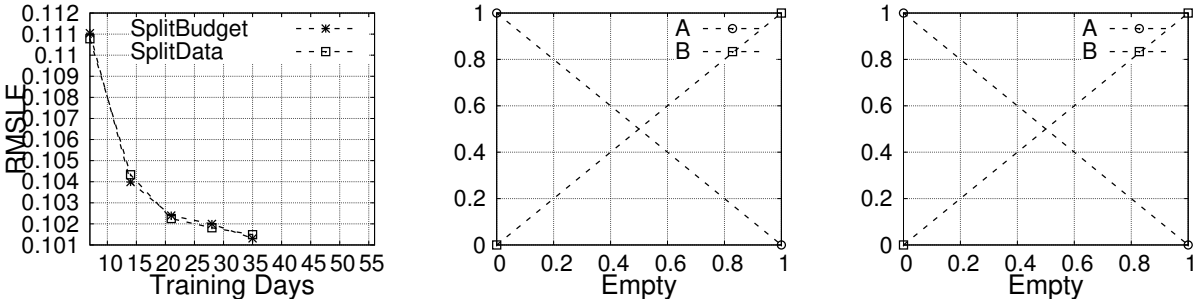
Datasets. We instantiate Scenario 3 on the Criteo ad click dataset [**?**]. It consists of 39 million ad impressions over seven days, each with 39 features, including IDs for user, the page the user is viewing, the ad shown, and whether the user has clicked on that ad. Of the 39 features, 26 are categorical and 13 are integers. A subsampled version of the dataset was the subject of a Kaggle competition to predict whether a user will click on an ad [**?**]. We use the same task and the winning predictive model as a baseline: a neural network binary classifier.

Method. Nine of Criteo's 26 categorical features are high-dimensional (10K or more values). We thus hypothesize that count featurization [2] will help improve the rate of learning for this predictive task and therefore will require models less raw data to converge. This could in turn support our scenario. We therefore register a count featurization



(a) **Tree Profile** (b) **Clustering Profile** (c) **Count Featurization Profile**

Fig. 7: **PTK Profiles.** XXX PLACEHOLDER IMAGE XXX This will have privacy budgets on the X axis and the amount of data required to achieve τ_1, τ_2, and τ_3 on the y axis.



(a) **Allocation Model Performance** (b) **Allocation Solving Time** (c) **PTK Prediction Latency**

Fig. 8: **PTK Performance.** XXX PLACEHOLDER IMAGE XXX

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.

Results. Fig. 6(a) shows ... Thus, in this scenario, PTKs are a useful unit for long-term retention and obviate the need for the primary task to retain access to – and therefore expose – raw data long-term.

6.1.4 Further Retention Use Cases

Datasets. We also instantiate retention use cases akin to Scenario 3 on the Douban and Taxi datasets. Our goal is to evaluate how much accuracy primary tasks defined on these datasets would lose if they reduced the amount of raw data they were given access at any time, and instead used PTKs to retain some information from the past.

Method. For Douban, we train User Embedding PTKs separately on the movie, music, and book datasets, using the complete datasets each time. Without sharing embeddings, we train recommenders for each of the three datasets, using increasing fractions of these datasets. For Taxi, we train historical statistics, clusters, and XXX datasets using the complete dataset. We use XXX task descriptions.

Results. Fig. 6(c) and 6(b) show ... Thus, xxx some conclusion here.

6.2 Resource Allocation Evaluation (Q2)

Budget/Data Allocation under Contention. Figure 8(a) will show the performance of the Taxi duration prediction model with 3 lines under the three different allocation models: evenly divide budget, evenly divide data, and the profiling based allocation. The Y axis will be model performance and the x axis will either be the total number of

PTKs or the number of PTKs in addition to those used by the model. We can x values of 50, 100, 1000, 10000. I don't think that we should talk about TFX because we never mentioned it before and it'd be confusing to introduce it here.

We exepct to see the performance degrade as we increase the number of PTKs but the profile based allocation should degrade less badly.

PTK Profiles. Figure 7 will show the performance profile of the tree PTK **??** and the clustering PTK **??** on the taxi dataset, and the count featurization ptk **??** on the criteo dataset. It may be better to include three from the same dataset but we can decide later exactly which figures we include here based on which are interesting. **[Instead of clustering, it would be better to use PCA or something for which we have a profiling approach. —RG]** xxx

The X axis will be privacy budget and the Y axis will be the amount of data required to meet the performance goal. There will be three performance goals: τ_1, τ_2, and τ_3. There will be two lines for each performance goal. The predicted amount of data to reach τ and the actual amount.

What the profiles look like for various PTKs and at various τ values. Compare to the experimental version of determining the profile: i.e., determine experimentally (e.g., using a binary search algorithm) how much data you need to reach τ.

Amounts of Data Used by Allocation. Plot or report the amount of data required to allocate all of the PTKs using our allocation procedure (sum over all partitions). You can place these numbers on top of the corresponding bars. In this little paragraph, refer to these numbers and comment that they are "reasonable."

6.3 Performance Evaluation (Q3)

Figure 8(c) show the prediction latency of two models Taxi duration prediction and citibike in the context only and the ptk configurations. The X axis will be percentile and the Y axis will show the latency. We expect that both of the context only models will lowest tail latency. I expect both of the PTK models will have a much higher tail latency. The CitiBike model because random forests often have a long tail latency and the taxi models because it will require two rounds of featurization. Both will still probably be dominated by network time.

Figure 8(b) will show the time required to solve the budget allocation problem after we have generated the profiles for each of the PTKs. The X axis will be the number of PTKs and the Y axis will be the time required to solve the MILP problem. We expect the time to polynomialy in the number of PTKs for which we're solving. Since this is under something like a day it should be fine since this will be run infrequently.

7 Analysis

[NOTES ONLY.] xxx

Need a security analysis here to clarify the threats that PTKs leave open and re-enforce why we believe we still add benefits.

Return to Multics principles and see how we redeem not only least privilege but also others.

Threats:

1. Ecosystem-wide threats: - A malicious employee will be able to access every data stream, state and service to which he is granted access. They can monitor the raw data, the internal state of servers, and the service's predictions to the users. However, if these resources are correctly siloed from employees who do not need access to them, then he won't be able to access them. The PTK abstraction aims to constrain the reach of any individual employee to the minimum they need, and assumes that access controls and firewalls correctly implement minimal access policies.

- An intruder into the ML ecosystem can, in the limit, retrieve arbitrary state in all of the predictive tasks/teams. This includes the state that Sage maintains, plus The PTK abstraction aims to constrain the reach of an intruder by allowing individual primary tasks to reduce their access retention periods by leveraging historical PTKs. Hence, the intruder will be able to retrieve data in the maximum retention period for all primary tasks for a particular data stream.

- Many attacks against integrity/availability: - Malicious employee registers a lot of PTKs with high eps. Won't get Sage to increase privacy budget beyond $\mathcal{E}$, so confidentiality wise we are OK. But it may get Sage to increase $\mathcal{T}$, at least to some point. We envision that the Sage admin will bound the max T they wish to enforce. The attacker may also cause extreme contention and hence new PTKs may be rejected by Sage. - Train bogus PTK, change a good PTK into worse. That sabotages the workloads that rely on the PTK.

2. Sage threats: Fewer, because Sage enforces a strict semantic. But there are a few caveats. - An intruder who breaks into the Sage service may change $\mathcal{E}$ or the bound on T. This will get Sage to weaken its guarantees. We assume that such changes are not possible without the (trusted) administrator's knowledge.

- When PTKs are registered, they are assumed to already have been designed. That design is usually done through a lot of experimentation and optimization on a dataset. The PTK's training procedure, can (in theory) reveal information about the dataset the developer used to design the PTK. We believe this is reasonable, but we invite care into how that is done and awareness XXX.

- A bad (or malicious) programmer may break the assumptions we make about the PTK functions, e.g., that `ptk.train_dp` and `ptk.profile_dp` satisfy the specified DP semantic for all of their outputs and side-effects. This can lead to leakage of user data the through the PTK. A natural solution would be to require that PTK functions must be certified through code review to be accepted by Sage.

The difference between Sage and the rest of the ecosystem is that Sage*enforces* the minimal-exposure policy while the rest of the ecosystem is merely encouraged to do so by leveraging Sage's abstractions. However, we acknowledge that enforcing minimal-exposure is non-trivial even with the PTK abstractions: when data is expired, all of its traces must be securely erased, all non-DP models based on them must be retrained without it, and worst of all; to determine, one needs to profile his model's dependency on the amount of data, akin to how we do it, and tune that with changes in the data.

Sage provides a few of these services, and could in theory be used to secure predictive models, too. They could develop DP versions of their predictive models and register them with Sage for periodic profiling, training, and releasing. This is useful to gain DP under continual output observation for the predictive tasks/teams. To get pan privacy for these tasks/teams, that's much harder, because it most likely means that once they register the predictive PTK, the engineers should no longer have access to the data. This means they won't be able to design and bootstrap new, improved versions of the predictive models, which is an important part of engineers' duties in an ML ecosystem. Designing an ecosystem-wide, $\mathcal{T}, \mathcal{E}$-pan private system is very much an open problem.

8 Related Work

Alternative Protection Models. §**??** discusses some alternative protection models for ML systems. Additional approaches researched in the literature follow. Local privacy is a strong data protection model based on differential privacy or randomized response, where the data is randomized *before* collection. The model is being used in practice [**?**] but only supports **[simple aggregate statistics]** and *not* general ML [**?**]. Federated learning XXX. xxx

Relationship with Standard DP Semantics. Sage's threat model and semantics relate closely to standard models based on differential privacy. Sage's global privacy semantic satisfies $\mathcal{E}$-DP under continual observation, a DP semantic defined for streaming computations, which requires the $\mathcal{E}$-DP guarantee to be met by all the externally visible outputs or side effects, combined []. Sage's minimal-exposure semantic relates to pan privacy, a DP semantic that seeks to protect a (streaming) DP computation from intruders by requiring that the $\mathcal{E}$-DP guarantee to be met not only by the outputs but also by the internal states of a DP computation, all combined. Sage's semantic, which permits bounded leakage, is strictly weaker than pan privacy, which permits no leakage. Pan privacy is known to reduce to local privacy under threat models that permit repeated unannounced intrusions []. By allowing some leakage, Sage breaks this reduction and is able to train general ML models effectively; by bounding the timespan of the leakage, it provides a meaningful protection semantic to its system administrators.

Relationship with DP ML Literature. Mostly algorithm-level, less ecosystem-level DP systems, and no minimal-exposure system designs.

A relevant direction is privacy budget allocation, which has been studied profusely particularly in the context of *static databases*. **[TODO(roxana): Need to read a few of these papers.]** xxx

Sample Complexity Estimation in Statistics. **[TODO(daniel/kiran): Review existing approaches and make a clear statement of what's novel here (application + DP formulation?).]** xxx

Resource Allocation in Systems. An enormous body of work exists on resource allocation in systems []. Our approach is most similar to that SLA resource allocation [], in that we profile the estimated resources (samples) needed to reach a particular performance target (τ) for our workloads. In addition to minimizing resource consumption, many other considerations have been formulated in the past in this literature: ensuring fair allocation across workloads,

avoiding starvation for these workloads, ensuring that workloads meet specific deadlines, etc. We acknowledge that our resource allocation formulation is limited and does not take into account such factors. A close look at this literature will broaden the scope of our own allocation procedure.

9 Conclusion

References

[1] 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.

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

A PTK Profiling Proofs

B Sage Semantic Proofs