Packing Privacy Budget Efficiently

Pierre Tholoniat[1], Kelly Kostopoulou[1], Mosharaf Chowdhury[2], Asaf Cidon[1], Roxana Geambasu[1], Mathias Lécuyer[3], and Junfeng Yang[1]

[1]Columbia University, [2]University of Michigan, [3]University of British Columbia

December 27, 2022

Abstract

Machine learning (ML) models can leak information about users, and differential privacy (DP) provides a rigorous way to bound that leakage under a given budget. This DP budget can be regarded as a new type of compute resource in workloads of multiple ML models training on user data. Once it is used, the DP budget is forever consumed. Therefore, it is crucial to allocate it most efficiently to train as many models as possible. This paper presents the scheduler for privacy that optimizes for efficiency. We formulate privacy scheduling as a new type of multidimensional knapsack problem, called *privacy knapsack*, which maximizes DP budget efficiency. We show that privacy knapsack is NP-hard, hence practical algorithms are necessarily approximate. We develop an approximation algorithm for privacy knapsack, *DPK*, and evaluate it on microbenchmarks and on a new, synthetic private-ML workload we developed from the Alibaba ML cluster trace. We show that DPK: (1) often approaches the efficiency-optimal schedule, (2) consistently schedules more tasks compared to a state-of-the-art privacy scheduling algorithm that focused on fairness (1.3–1.7× in Alibaba, 1.0–2.6× in microbenchmarks), but (3) sacrifices some level of fairness for efficiency. Therefore, using DPK, DP ML operators should be able to train more models on the same amount of user data while offering the same privacy guarantee to their users.

1 Introduction

Machine learning (ML) workloads are consuming an essential and scarce resource – *user privacy* – but they are not always accounting for or bounding this consumption. A large company may train thousands of models over user data per week, continuously updating these models as it collects new data. Some of the models may be released to mobile devices or distributed globally to speed up prediction. Unfortunately, there is increasing evidence that ML models can reveal specific entries from their original training sets [6, 7, 13, 54], both through parameters and predictions, thereby potentially leaking user data to adversaries. Intuitively, the more one learns from aggregate user data, the more one should expect to also learn (and hence leak) about individual users whose data is used. This intuition has been proven for simple statistics [10] and appears to hold experimentally for ML models. Therefore, user privacy can be viewed as a *resource* that is consumed by tasks in an ML workload, and whose consumption should be accounted for and bounded to limit data leakage risk.

Differential privacy (DP) [12] provides a rigorous way to define the privacy resource, and to account for it across multiple computations or tasks, be they ML model training tasks or statistic calculations (§2). DP randomizes a computation over a dataset (e.g. training an ML model or computing a statistic) to bound the leakage of entries in the dataset through the output of the computation. Each DP computation increases this bound on data leakage, consuming some of the data's *privacy budget*, a pre-set quantity that should never be exceeded to maintain the privacy guarantee. In workloads with a large number of tasks that continuously train models on a private corpus or stream, the data's privacy budget is a very scarce resource that must be efficiently allocated to enable the execution of as many tasks as possible.

Recent work [38, 40, 59] has explored how to expose data privacy to tasks as a new *computing resource* (§2). Because the privacy resource behaves differently from traditional computing resources – such as CPU, GPU, RAM, etc. – scheduling it requires new algorithms, or adaptations of existing ones to the privacy resource. The state-of-the-art algorithm for privacy scheduling, DPF [40], optimizes for fairness, providing guarantees that are derived from traditional max-min fairness. Unfortunately, as is often the case in scheduling [8, 22, 23, 28, 33, 50], fairness can come at the cost of allocation efficiency. For privacy, we find that this inefficiency is especially evident in workloads that exhibit a high degree of *heterogeneity* either in the data segments they request (e.g. a workload containing tasks that run on data collected from different time ranges), or in the types of tasks they contain (e.g. a workload mixing different types of statistics and ML algorithms). In such cases, we show that a scheduler that optimizes for efficiency rather than fairness can schedule up to 2.6× more tasks for the same privacy budget!

In this paper, we explore the first practical *efficiency-oriented privacy schedulers*, which aim to maximize the number of scheduled tasks, or the total utility of scheduled tasks if tasks are assigned utility weights (§3). We first introduce a new formulation of the DP scheduling problem, which optimizes for efficiency, and show that it maps to the NP-hard multidimensional knapsack problem, requiring practical approximations to solve in practice. We demonstrate that prior work, which optimizes for fairness, can be seen as an inefficient heuristic to solve this problem, and how a better heuristic for multidimensional knapsack yields more efficient DP scheduling. We then show that instantiating the privacy scheduling problem to Rényi DP (RDP) accounting, a state-of-the-art, efficient DP accounting mechanism, introduces a new dimension with unusual semantics to the scheduling problem. To support this new dimension, we define a new knapsack problem that we call the *privacy knapsack*, which we show is also NP-hard. Finally, we propose a new RDP-aware heuristic for the privacy knapsack, instantiate it into a new scheduling algorithm called *DPK*, provide a formal analysis of its approximation properties, and discuss when one should expect to see significant efficiency gains from it (§4).

We implement DPK in a Kubernetes-based orchestrator for data privacy [40] and an easily-configurable simulator (§5). Using both microbenchmarks and a new, synthetic, DP-ML workload we developed from the Alibaba's ML cluster trace [58], we compare DPK to DPF, the optimal privacy knapsack solver, and first-come-first-serve (FCFS) (§6). DPK schedules significantly more tasks than DPF (1.3–1.7× in Alibaba and 1.0–2.6× in microbenchmarks), and closely tracks the optimal solution, at least up to a small number of blocks and tasks where it is feasible for us to obtain the optimal solution. DPK on Kubernetes can scale to thousands of tasks, and incurs a relatively modest scheduler runtime overhead. Still, by focusing on efficiency, DPK sacrifices some level of fairness compared to DPF: in the Alibaba workload, DPF is able to schedule 90% of tasks that request less or equal than their privacy budget “fair-share”, while DPK schedules only 60% of such tasks. This is inevitable given the rather fundamental tradeoff between efficiency and fairness in scheduling. Our work thus fills in an important gap on algorithms that prioritize efficiency over fairness, as we believe will be desirable given the scarcity of this essential new resource in ML systems, user privacy.

2 Background

2.1 Threat Model

We are concerned with the most commonly used threat model in DP: the *central model*. We assume a trusted curator that stores all sensitive data in a central location and trust that they will not directly inspect or share it. We also trust the tasks that are submitted by the users and expect those to satisfy a particular (ε,δ)-DP guarantee. However, we do not trust the recipients of results released by the system, or the locations in which they are stored. Access to those results may expose sensitive information about individuals, and we impose DP guarantees across all the processes that generate them. Released results may be statistical aggregates, ML model predictions, or entire ML models. Related work shows that these can be leveraged to perform malicious acts. *Membership inference* attacks [3, 14, 30, 54] allow the adversary to infer whether an individual is in the data used to generate the output. *Data reconstruction attacks* [6, 10, 13] allow the adversary to infer sensitive attributes about individuals that exist in this data. We tackle both types of attack.

Our focus is not on singular models or statistics, released once, but rather on *workloads of many models or statistics*, trained or updated periodically over windows of data from user streams. For example, a company may train an autocomplete model daily or weekly to incorporate new data from an email stream, distributing the updated models to mobile devices for fast prediction. Moreover, the company may use the same email stream to periodically train and disseminate multiple types of models, for example for recommendations, spam detection, and ad targeting. This creates ample opportunities for an adversary to collect models and perform privacy attacks to siphon personal data. To prevent such attacks, our goal is to *maintain a global* $(\varepsilon^G, \delta^G)$*-DP guarantee over the entire workload consisting of many tasks*.

2.2 DP Background

Because our scheduling algorithms intimately leverage properties of DP theory to achieve good efficiency, we include a detailed DP background. To begin with, DP is known to address both membership inference and data reconstruction attacks [6, 13, 32, 54]. Intuitively, both attacks work by finding data points (which can range from individual events to entire users) that make the observed model more likely: if those points were in the training set, the likelihood of the observed model increases. DP prevents these attacks by ensuring that no specific data point can drastically increase the likelihood of the model produced by the training procedure.

DP randomizes a computation over a dataset (e.g., the training of an ML model) to bound a quantity called *privacy loss*, defined as a measure of the change in the distribution over the outputs of the randomized algorithm incurred when a single data point is added to or removed from the input dataset. Privacy loss is a formalization of what one might colloquially call “leakage” through a model. Formally, given a randomized algorithm, $\mathcal{A} : \mathcal{D} \to \mathcal{Y}$, for any datasets $\mathcal{D}, \mathcal{D}'$ that differ in one entry and for any output $y \in \mathcal{Y}$:

$$\text{PrivacyLoss}(y, \mathcal{D}, \mathcal{D}') = \log\left(\frac{P(\mathcal{A}(\mathcal{D}) = y)}{P(\mathcal{A}(\mathcal{D}') = y)}\right).$$

Satisfying pure ε-DP [12] and its relaxed (ε,δ)-DP version [11] means bounding the privacy loss by some fixed, parameterized value, $\varepsilon > 0$, called *privacy budget*, potentially with high probability $1-\delta$, for $\delta \in [0,1)$. Specifically, it means that for any y, $\mathcal{D}$, $\mathcal{D}'$ as above, the following holds with probability at least $1-\delta$ when $y \sim \mathcal{A}(\mathcal{D})$ or $y \sim \mathcal{A}(\mathcal{D}')$:

$$|\text{PrivacyLoss}(y, \mathcal{D}, \mathcal{D}')| \leq \varepsilon.$$

There are multiple mechanisms to transform a (non-randomized) computation – such as computing a statistic or performing a gradient step – into a DP computation. At a high level, if the computation has (potentially multidimensional) real-valued output, one adds noise from specific distributions, such as Gaussian or Laplace, to the computation's output. Through properties of these well-known distributions, one can then compute an upper bound on the privacy loss.

These traditional DP definitions have the strength of being relatively interpretable: for a small value of ε (e.g., $\varepsilon \leq 1$), ε-DP can be interpreted as a guarantee that an attacker who inspects the output of an ε-DP computation will not learn anything new with confidence about any entry in the training set that they would not otherwise learn if the entry were not in the training set [57]. Similarly, for small δ (e.g., $\delta < \frac{1}{n^2}$ for dataset size n), (ε, δ)-DP guarantee is roughly a high-probability ε-DP guarantee. The advantage of (ε, δ)-DP is support for a richer set of randomization mechanisms, such as adding noise from a Gaussian distribution, which pure DP cannot, and which often provide better privacy-utility tradeoffs. That is why (ε, δ)-DP is the reference privacy definition for DP ML.

Basic DP Accounting. A key property of DP is *composition*, the ability to compute the overall DP guarantee under multiple DP tasks. The most basic composition property offers a simple and convenient arithmetic: the composition of two $(\varepsilon_1, \delta_1)$-DP and $(\varepsilon_2, \delta_2)$-DP tasks is $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-DP. In practice, the abstraction used for DP composition is a *privacy accountant* [43], which keeps track of all DP tasks, composes their privacy budget, and potentially prevents further data access when a preset, global bound of the privacy loss is reached. A DP accountant based on basic composition will sum the ε's and δ's of all DP tasks running on the data, leading to the global DP guarantee degrading linearly with the number of tasks m.

Rényi DP Accounting ((α, ε)-RDP). RDP [44] is a recent DP definition that defines the privacy loss differently, sacrificing interpretability for tighter analysis of randomization mechanisms and how they compose with each other. It yields even better privacy-utility tradeoffs, especially for DP ML, and hence it has been adopted inside the privacy accountants of most DP ML platforms [15, 19, 20]. Instead of defining the privacy loss based on probability ratios as traditional DP does, RDP defines it in terms of the Rényi divergence, a particular distance between the distributions over all possible outcomes for $\mathcal{A}(\mathcal{D})$ and $\mathcal{A}(\mathcal{D}')$. Rényi divergence has a parameter, $\alpha > 1$, called *order*:

$$\text{PrivacyLoss}_\alpha(\mathcal{D}, \mathcal{D}') = \frac{1}{\alpha - 1} \log \mathop{\mathbb{E}}_{y \sim \mathcal{A}(\mathcal{D})} \left(\frac{P(\mathcal{A}(\mathcal{D}) = y)}{P(\mathcal{A}(\mathcal{D}') = y)} \right)^\alpha.$$

As before, (α, ε)-RDP requires that $|\text{PrivacyLoss}_\alpha(\mathcal{D}, \mathcal{D}')| \leq \varepsilon$ for any datasets $\mathcal{D}, \mathcal{D}'$ differing in one entry. Known noise mechanisms for traditional DP, such as those adding Gaussian or Laplace noise to the computation's output, have known RDP curves ensuring a bound $\varepsilon(\alpha)$ for any α value.

RDP is less interpretable than traditional DP due to the complexity of Rényi divergence. However, one can always translate from (ε, α)-RDP to $(\varepsilon_{DP}, \delta)$-DP [44] for any appropriately ranged values of α, ε, and δ:

$$\varepsilon_{DP} = \varepsilon + \frac{\log(1/\delta)}{\alpha - 1}. \quad (1)$$

RDP's greatest advantage over traditional DP – and the reason for its recent adoption by most major DP ML platforms as well as for our special consideration of it in this paper – is its support for *both efficient and convenient composition*. In RDP, the composition of two (ε_1, α)-RDP and (ε_2, α)-RDP mechanisms is $(\varepsilon_1 + \varepsilon_2, \alpha)$-RDP, a similarly convenient arithmetic as for basic composition in traditional DP. However, whereas with traditional DP, composing m mechanisms degrades the global guarantee linearly with m, with RDP, the global guarantee degrades with $\sqrt{m}$ when applying composition followed by conversion to traditional DP through Eq. 1. This means that RDP can allow composition of more DP computations with the same ε guarantees; the advantage is particularly significant with a large m.

Since popular DP ML algorithms, such as DP SGD, consist of tens of thousands iterations of the same rudimentary DP computation (computing one gradient step over a sample batch), they require the most efficient composition accounting method. This is why most DP ML platforms have adopted privacy accountants that internally operate on RDP to compose across steps and then translate the cumulative RDP guarantee into traditional DP (with Eq. 1) to provide an interpretable privacy semantic externally. Similarly, since our goal is to develop *efficient scheduling algorithms* – that pack as many DP ML tasks as possible onto a fixed privacy budget – it is impending upon us to consider RDP accounting in our scheduling formulations.[1] We do so in a similar way: internally, some of the algorithms we propose use RDP accounting (albeit to compose across ML training tasks, *not* across gradient steps within a task) but externally we will always relate a traditional DP guarantee. As it turns out, operating on RDP internally creates interesting challenges for scheduling, about which we discuss in §3.2.

2.3 Privacy Scheduling Background

The preceding DP accounting view of composition implies that a global DP guarantee can be seen as a finite privacy budget to be allocated to tasks running on the data. In practice, many DP systems maintain separate privacy budgets for different subsets of the dataset. Indeed, a property of DP called *parallel composition* [43] guarantees that, for two disjoint subsets of the input domain D_1 and D_2, the result of an ε_1-DP computation on D_1 and an ε_2-DP computation on D_2 is $\max(\varepsilon_1, \varepsilon_2)$-DP. This property is crucial for practical DP applications that handle many computations on non-overlapping parts of the dataset, such as SQL databases [43, 59], analytics frameworks [5, 63] or ML platforms for continuous training [38, 40].

A recent line of work [38, 40] has argued for the global privacy budget to be managed as a new type of computing resource in workloads operating on user data: its use should be

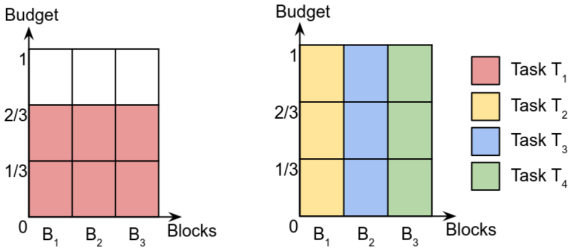
[1] We considered, and discarded, advanced composition for traditional DP, which is also efficient but involves untenable arithmetic [46].

tracked and carefully allocated to competing tasks. We adopt the same focus on ML platforms for continuous training on user data streams, such as Tensorflow-Extended (TFX), and build on the same basic operational model [38] and key abstractions and algorithms [40] for monitoring and allocating privacy in DP versions of these platforms. The operational model is as follows. Similar to TFX, the user data stream is split into multiple non-overlapping *blocks* (called *spans* in TFX [56]), for example by time, with new blocks being added over time. Blocks can also correspond to partitions given by SQL `GROUP BY` statements over public keys, such as in Google's DP SQL system [59] or in the DP library used for the U.S. Census [5]. There are multiple tasks, dynamically arriving over time, that request to compute (e.g., train ML models) on subsets of the blocks, such as the most recent N blocks. The company owning the data wants to enforce a global traditional DP guarantee, $(\varepsilon^G, \delta^G)$-DP, that cannot be exceeded across these tasks. Each data block is associated with a global privacy budget (fixed a priori), which is consumed as DP tasks compute on that block until it is depleted.

Luo et al. [40] incorporated *privacy blocks*, i.e., data blocks with privacy budget, as a new compute resource into Kubernetes, to allocate privacy budget from these blocks to tasks that request them. To request privacy budget from a privacy block, a task i sets a demand vector (d_i) of length m, equal to the number of blocks in the system. The demand vector specifies the privacy budget that task i requests for each individual block in the system (with a zero demand for blocks that it is not requesting). If task i is allocated, then its demand vector is consumed from the blocks' privacy budgets. When a block's privacy budget reaches zero, no more tasks can be allocated for that block and the block is removed. This ensures that a block of user data will not be used to extract so much information that it risks leaking information about the users. In this sense, each privacy block is a *non-replenishable* or finite resource. It is therefore important to carefully allocate budget from privacy blocks across tasks, so as to pack as many tasks as possible onto the blocks available at any time. That's the goal of our *efficiency-oriented privacy scheduling*.

3 Efficiency-Oriented Privacy Scheduling

We introduce a new formulation of the DP scheduling problem, which optimizes for efficiency. We first develop an *offline* version of this problem, in which the entire workload is assumed to be fixed and known a priori, and study efficient DP scheduling with basic DP accounting (§3.1). We show that offline DP scheduling maps to the NP-hard multidimensional knapsack problem, requiring practical approximations to solve in practice. We show that DPF [40], a previous DP scheduler that optimizes for fairness, can be seen as an inefficient heuristic for this problem, and how a better heuristic yields more efficient DP scheduling with multiple data blocks. We then show that RDP accounting introduces a new dimension with unusual semantics to the scheduling problem (§3.2). To support this new dimension, we define a new knapsack problem, called the *privacy knapsack*, which is also NP-hard. We propose an RDP-aware heuristic for the privacy knapsack, instantiate it a scheduling algorithm called *DPK*, and analyze its performance (§3.3). Finally, we show how we adapt this heuristic to the online case (§3.4).



(a) Inefficient allocation with DPF (b) Efficient allocation

Fig. 1: **Example of allocations with basic DP accounting.** Task T_1 requests privacy budget from 3 blocks, B_1, B_2, B_3. Tasks T_2, T_3, T_4 request slightly more privacy budget, but each one from one distinct block: B_1, B_2, B_3, respectively. In 1(a), DPF sorts these tasks based on their dominant shares: T_1 first (because its dominant share is lower, even though it demands budget from all the blocks), then T_2, T_3, T_4 in arbitrary order. After T_1 is scheduled, there is no more budget for other tasks. Meanwhile, in 1(b) an efficient scheduler can allocate 3 tasks.

3.1 Efficient Scheduling Under Basic DP Accounting

We define the *global efficiency* of a scheduling algorithm as either the number of scheduled tasks or, more generally, the sum of *weights* w_i of scheduled tasks, for cases when different tasks have different utilities (a.k.a. profits or weights) to the organization. When the goal is to optimize global efficiency, we can model privacy budget scheduling in a multi-block system such as TFX as a multidimensional knapsack problem.

Consider a fixed number of n tasks $(t_1, \ldots, t_n)$ that need to be scheduled over m blocks, each with c_j remaining capacity. Each task has a demand vector d_{ij}, which represents the demand by task i for block j, and a weight w_i if it is successfully scheduled (when w_i is equal across all tasks, the problem is to maximize the number of scheduled tasks).ß We can formulate this problem as the standard multidimensional knapsack problem [35], where x_i are binary variables:

$$\max_{x_i \in \{0,1\}} \quad \sum_{i=1}^{n} w_i x_i$$

$$\text{subject to} \quad \forall j \in [m] : \sum_{i=1}^{n} d_{ij} x_i \leq c_j .$$

W.l.o.g., we assume there is not enough budget to schedule all tasks: $\forall j \in [m] : \sum_{i=1}^{n} d_{ij} > c_j$. Otherwise, the knapsack problem is trivial to solve. If some blocks have enough budget but not others, we can set the blocks with enough budget aside, solve the problem only on the blocks with contention, and incorporate the remaining blocks at the end.

The Need for Heuristics. The multidimensional knapsack problem is known to be NP-hard [35], so DP scheduling cannot be solved exactly, even in the offline case. There exist some general-purpose polynomial approximations for this problem, but they are exponential in the approximation parameter and become prohibitive for large numbers of dimensions (for us, many blocks). In §6.2, we show that the Gurobi [27] solver quickly becomes intractable with just 7 blocks.

The standard approach to practically solve knapsack problems is to develop specialized approximations for a specific domain of the problem, typically using a *greedy algorithm* of the following form [35]: (1) sort tasks according to a *task efficiency metric* (denoted e_i) and (2) allocate tasks in order, starting from the highest-efficiency tasks, until the algorithm cannot pack any new tasks. In such algorithms, the main challenge is coming up with good task efficiency metrics that leverage domain characteristics to meaningfully approximate the optimal solution while remaining practical.

Inefficiencies Under a Fair Scheduling Heuristic. Interestingly, we can model DPF [40], the state-of-the-art privacy scheduling algorithm, as a greedy heuristic for privacy knapsack, albeit one that offers no efficiency guarantee. DPF schedules tasks with the smallest dominant share ($\max_j \frac{d_{ij}}{c_j}$) first. Folding in task weights, this becomes equivalent to a greedy algorithm with an efficiency metric defined as: $e_i := \frac{w_i}{\max_j \frac{d_{ij}}{c_j}}$. Unfortunately, under this efficiency metric DPF can stray arbitrarily far from the optimal even in simple cases. The reason lies in the maxima over j, which is crucial to ensure the fair distribution of DP budget, but causes DPF to ignore multidimensionality in data blocks. Fig. 1 gives an example using traditional DP and a workload of 4 tasks. DPF sorts tasks by dominant share and schedules only one task. Meanwhile, a better efficiency metric would consider the "area" of a task's demand, thereby sorting tasks T_2, T_3 and T_4 before T_1, resulting in 3 tasks getting scheduled.

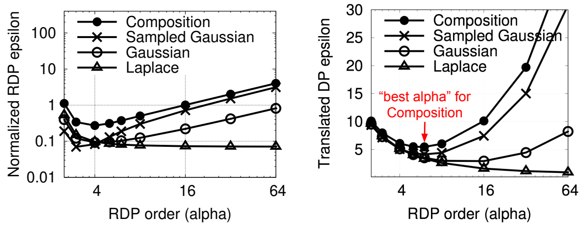
Area-Based Metric for Efficient Scheduling Over Blocks. We take inspiration from single-dimensional knapsacks, in which the efficiency e_i of task i is usually defined as the task's weight-to-demand ratio: $e_i := w_i/d_i$. A natural extension to multiple blocks uses a known heuristic for multidimensional knapsacks [49] to capture the entire demand of a task:

$$e_i := \frac{w_i}{\sum_j \frac{d_{ij}}{c_j}}, \tag{2}$$

where $\frac{d_{ij}}{c_j}$ is task i's DP budget demand for block j, normalized by the remaining capacity of block j. This normalization is important to express the scarcity of a demanded resource. Unlike the DPF fair scheduling metric, Eq. 2 considers the entire "area" of a task's demand to compute its efficiency, addressing the inefficiency from Fig. 1. A task requesting a large budget across blocks is not scheduled even if its demand on any block (dominant share) is small.

3.2 Efficient Scheduling Under RDP Accounting

The above heuristic is satisfactory for traditional DP accounting, but practitioners and state-of-the-art ML algorithms use the more efficient RDP accounting (§2). Recall that with RDP, multiple bounds on the privacy loss can be computed, for various RDP orders α (Eq. 2.2). This yields an *RDP order curve* $\varepsilon(\alpha)$ for that computation. For instance, adding noise from a Gaussian with standard deviation σ into a computation results in $\varepsilon(\alpha) = \frac{\alpha}{2\sigma^2}$. Other mechanisms, such as subsampled Gaussian (used in DP-SGD) or Laplace (used in simple statistics), induce other RDP curves. These curves are highly non-linear and their shapes differ from each other. This makes it difficult to know analytically what the privacy loss function will look like when composing multiple computations with heterogeneous RDP curves. For this reason, typically the RDP ε bound is computed on a few discrete α values ($\{1.5, 1.75, 2, 2.5, 3, 4, 5, 6, 8, 16, 32, 64\}$ [44]), on which the composition is performed. Importantly composition for each α is still additive, a key element of RDP's practicality.



(a) RDP curves

(b) DP translation

Fig. 2: **Example RDP curves and DP translation.** (a) RDP curves for Gaussian, subsampled Gaussian, and Laplace mechanisms, each with std-dev $\sigma = 2$, plus their composition. (b) Translation to $(\varepsilon_{DP}, 10^{-6})$-DP. The "best" (i.e., tightest) alpha differs among mechanisms. For composition, best is $\alpha = 6$, giving $\varepsilon_{DP} = 5.5$.

Fig. 2(a) shows RDP curves for three example computations, each using a popular DP mechanism: the Gaussian corresponds to a multidimensional statistic (a histogram); the subsampled Gaussian corresponds to DP-SGD training; and Laplace corresponds to a simple statistic (an average). Different computations exhibit different RDP curves, with different orderings of the Rényi divergence bound at different α's. The subsampled Gaussian is tighter at lower α values; the Laplace is tighter for large α's. The figure also shows the RDP curve for the composition of the three computations.

Fig. 2(b) shows the translation of these four curves into traditional DP (using Eq. 1). For each computation, any value of $\alpha > 1$ will translate into a different traditional ε. Some traditional ε translations are very loose, others are tighter, but they are all valid simultaneously. Because of this, we can pick the α that gives us the best traditional ε guarantee and disregard the rest as loose bounds. This *best alpha* differs from computation to computation: in our example, for the Gaussian it is $\alpha \approx 16$; for the subsampled Gaussian $\alpha \approx 6$; and for the Laplace $\alpha \geq 64$. The best alpha for the composition of all three computations is $\alpha \approx 6$, yielding $(\varepsilon = 5.5, \delta = 10^{-6})$-DP. If we were to analyze and compose the three computations directly in traditional DP, we would obtain a looser global guarantee of $(\varepsilon = 7.8, \delta = 10^{-6})$-DP. This gap grows fast with the number of computations. Herein lies RDP's power, but also a significant challenge when trying to allocate its budget across competing computations.

Notice that when translating from RDP to traditional DP with Eq. 1, one chooses the most advantageous α for the final traditional DP guarantee, ignoring all other RDP orders. This new α dimension therefore has a different semantic than the traditional multidimensional knapsack one. Indeed, the traditional knapsack dimension semantic is that an allocation has to be within budget *along all dimensions*. This is a good fit for our block dimension, as we saw in §3.1. Instead, an allocation is valid along the α dimension as long as the allocation is within budget for *at least one dimension*. This creates opportunities for efficient scheduling, as the allocator can go over-budget for all but one α order. It also creates a new challenge, as the α order that will yield the most efficient allocation is unknown a priori and depends on the chosen combination of tasks. Since the traditional multidimensional knapsack does not encode this new semantic, we define a new multidimensional knapsack problem for efficient RDP scheduling.

The Privacy Knapsack Problem. To accommodate RDP, we need to modify the standard multidimensional knapsack problem to support alpha orders for each block and task demand. We express the capacity as $c_{j\alpha}$ (the available capacity of block j on order α), each demand vector as $d_{ij\alpha}$ (the demand of task i on block j's order α), and require that the sum of the demands will be smaller or equal to the capacity for *at least one of the alpha orders*. Therefore, we formulate the privacy knapsack as follows:

$$\max_{x_i \in \{0,1\}} \qquad \sum_{i=1}^{n} w_i x_i$$

$$\text{subject to} \quad \forall j \in [m], \exists \alpha \in A : \sum_{i=1}^{n} d_{ij\alpha} x_i \leq c_{j\alpha}.$$

We prove three properties of privacy knapsack (proofs in Appendix §A):

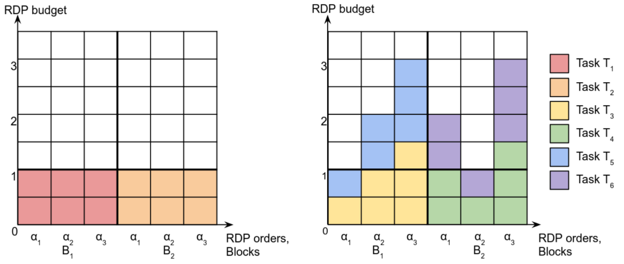
Property 1. The decision problem for the privacy knapsack problem is NP-hard.

Property 2. In the single-block case, there is a fully polynomial time approximation scheme (FPTAS) for privacy knapsack. I.e., with $w^{\max}$ the highest possible global efficiency, for any $\eta > 0$ we can find an allocation with global efficiency $\hat{w}$ such that $w^{\max} \leq (1+\eta)\hat{w}$, with a running time polynomial in n and $1/\eta$.

Property 3. For $m \geq 2$ blocks, there is no FPTAS for the privacy knapsack problem unless P=NP.

While Prop. 1 and 3 are disheartening (though perhaps unsurprising), Prop. 2 gives a glimmer of hope that at least for single-block instances, we can solve the problem tractably. Indeed, as we shall see, this property is crucial for our solution.

Fair Scheduling With Multiple RDP Alpha Orders. Interestingly, fair scheduling with DPF for RDP can once again be expressed as an ordering heuristic for the privacy knapsack, in which efficiency is defined as $e_i := \frac{w_i}{\max_{j\alpha} \frac{d_{ij\alpha}}{c_{j\alpha}}}$. However, this approach is even more inefficient than under traditional DP.



(a) Inefficient allocation with DPF

(b) Efficient allocation

Fig. 3: **Example of allocations with RDP accounting.** In Fig. 3(a), DPF treats RDP orders like a regular resource and orders tasks by dominant share, allocating only 2 tasks in this example. Meanwhile, Fig. 3(b) leverages the fact that only one order per block has to be below the capacity (here, α_1 for block B_1 and α_2 for block B_2). Tasks T_3 and T_5 have a large dominant share of 1.5 but are efficient because they request only 0.5 for B_1's best alpha, α_1.

In addition to the previous multi-block inefficiency (§3.1), this fair scheduling approach exhibits a new inefficiency under RDP, regardless of the number of blocks it is invoked on (e.g., even if applied to non-block-based DP systems, such as DP SQL databases). Fig. 3 gives an example using two blocks and a workload of 6 tasks, each requesting only 1 block. In Fig. 3(a), DPF sorts tasks by the highest demands across all α's and allocates only 2 tasks. A better efficiency metric would sort tasks by demands at the α value that can pack the most tasks (a.k.a., best alpha for composition), ultimately scheduling 4 tasks in Fig. 3(b). Note that the best alpha is not necessarily the same for each block.

We conclude that an efficiency metric that simply takes the maximum of the dominant shares is neither efficient for scheduling multiple privacy blocks, nor for scheduling privacy budget in systems that use RDP accounting. However, a direct extension of our "area based" efficiency metric in Eq. 2 does not appropriately handle RDP alpha orders either, as it does not account for the specific semantic of the α order. We next describe our new efficiency metric, that is optimized for efficiently scheduling tasks across multiple blocks and supports RDP.

3.3 DPK Algorithm

Intuitively, to support the "at least one" semantic of the α order from RDP, we need an efficiency metric that makes it less attractive to pack a task that consumes a lot of budget at what will ultimately be the *best alpha*, defined as the RDP order that packs the most tasks (or the most weight) while remaining under budget. That best alpha is ultimately the only one for which the demands of tasks matter and hence should be the one used for computing an efficiency metric. The challenge is that for workloads consisting of tasks with heterogeneous RDP curves, the best alpha is not known a

Algorithm 1 DPK Offline Algorithm

```
global variables
    tasks $i$, blocks $j$, RDP orders $\alpha$ capacities $c_{j\alpha}$
    approximation factor $\eta$, demands $d_{ij\alpha}$, weights $w_i$
function COMPUTEBESTALPHA(block $j$)
    for $\forall\alpha$ do
        ${w^{\hat{max}}}_{j\alpha} \leftarrow$ SINGLEBLOCKKNAPSACK($c_\alpha, d_{ij\alpha}, w_i, \frac{2}{3}\eta$)
    return $\arg\max_\alpha {w^{\hat{max}}}_{j\alpha}$
function COMPUTEEFFICIENCY(task $i$, best alphas $\hat{\alpha}_j^{max}$)
    return $w_i / \sum_j (d_{ij\hat{\alpha}_j^{max}} / c_{j\hat{\alpha}_j^{max}})$
function CANRUN(task $i$)
    return $\forall j, \exists\alpha : \sum_{i'=1}^{i} d_{i'j\alpha} \leq c_{j\alpha}$
function SCHEDULE(tasks $i$)
    for $\forall j$ do
        $\hat{\alpha}_j^{max} \leftarrow$ COMPUTEBESTALPHA$(c_{j\alpha}, d_{ij\alpha}, w_i)$
    sorted_tasks $\leftarrow$ tasks.sortBy(COMPUTEEFFICIENCY($\hat{\alpha}_j^{max}$))
    for $i$ in sorted_tasks do
        if CANRUN($d_{ij\alpha}$) then
            Run task $i$, consuming the demanded budget
```

priori. Our idea is to compute it for the given set of RDP curves constituting the workload (recall that we are in the offline case), and to focus a task's efficiency metric on that best alpha as the only relevant dimension.

Recall from Prop. 2 that in the single-block case, we can solve privacy knapsack with polynomial-time η-approximation for arbitrarily small $\eta > 0$. This means we can solve a single-block knapsack problem *separately for each block* j that determines the best alpha that will pack the most tasks (or maximal weight) among tasks requesting block j, taking only their request for that block into account. We define the maximum utility for block j and order α as $w_{j\alpha}^{max} := \max_{x_i} \sum_{i:d_{ij\alpha}>0}^{n} x_i w_i$ subject to $\sum_i x_i d_{ij\alpha} \leq c_{j\alpha}$. We take $\hat{w}_{j\alpha}^{max}$ a $\frac{2}{3}\eta$-approximation of $w_{j\alpha}^{max}$ ($\frac{2}{3}$ is justified by proof below). Based on this, we define the efficiency of task i as:

$$e_i := \frac{w_i}{\sum_{j\alpha}\left(\frac{d_{ij\alpha}}{c_{j\alpha}} \text{ if } (\alpha == \arg\max_{\alpha'} \hat{w}_{j\alpha'}^{max}) \text{ else } 0\right)} \tag{3}$$

Alg. 1 shows *DPK*, our greedy approximation with the efficiency metric in Eq. 3. This algorithm addresses both of the problems we identified with DPF. Moreover, we show that the manner in which DPK handles RDP is not just better than DPF in particular, but rather has two important generally desirable properties. First, DPK reduces to the traditional multidimensional knapsack efficiency metric of Eq. 2 when only one α exists, e.g. for traditional DP:

Property 4. If the dimension of α values is one (e.g., with traditional DP), DPK reduces to the traditional multidimensional knapsack heuristic from Eq. 2.

Proof. With one dimension, $\alpha == \arg\max_{\alpha'} \hat{w}_{j\alpha'}^{max}$ is always true. □

Second, DPK is a *guaranteed approximation* of the optimal in the specific cases when such an approximation is possible, the single-block case:

Property 5. In the single-block case, DPK is a $(\frac{1}{2}+\eta)$-approximation algorithm for privacy knapsack.

Proof. Call $\hat{\alpha} \triangleq \arg\max_{\alpha'} \hat{w}_{j\alpha'}^{max}$. By construction we have $w_{j\hat{\alpha}}^{max} \leq (1+\frac{2}{3}\eta)\hat{w}_{j\hat{\alpha}}^{max}$. In the single-block (index j) case, Eq. 3 means that tasks are greedily allocated by decreasing $\frac{w_i}{d_{ij\hat{\alpha}}}$, a well known $1/2$-approximation to the one dimensional knapsack problem [35]. Hence, $w_{j\hat{\alpha}}^{max} \leq (1+\frac{2}{3}\eta)\hat{w}_{j\hat{\alpha}}^{max} \leq (1+\frac{2}{3}\eta)(1+\frac{1}{2})\sum_{i=1}^{n} x_i w_i = (1+\frac{1}{2}+\eta)\sum_{i=1}^{n} x_i w_i$. □

Because of Prop. 3, a similar multi-block efficiency guarantee cannot be formulated (for DPK as well as any other poly-time algorithm). However, §6 shows that in practice, DPK performs close to the optimal solution of privacy knapsack in terms of global efficiency, yet it is a computationally cheap alternative to that intractable optimal solution.

3.4 Adapting to the Online Case

In practice, new tasks and blocks arrive dynamically in a system such as TFX, motivating the need for an online scheduling algorithm. We adapt our offline algorithm to the online case by scheduling a batch of tasks on the set of available blocks every T units of time. To prevent expensive tasks from consuming all the budget prematurely, similar to DPF, we schedule each batch on a fraction of the total budget capacity: at each scheduling step we unlock an additional $1/N$ fraction of the block capacity. More precisely, at each scheduling time $t = kT$, we execute Alg. 1 on the tasks and blocks currently in the system, but we replace block j's capacity by:

$$c_{j\alpha}^{t} = \frac{\min(\lceil (t-t_j)/T \rceil, N)}{N}\varepsilon_{j\alpha} - \sum_{i' \in A_t} d_{i'j\alpha},$$

where $\varepsilon_{j\alpha}$ is the total capacity of block j (computed from Prop. 1), t_j is the arrival time of block j, $\lceil (t-t_j)/T \rceil$ is the number of scheduling steps the block has witnessed so far (including the current step), and A_t is the set of tasks previously allocated.

As with the offline algorithm, at the time of scheduling all the tasks are sorted by the scheduling algorithm. The scheduler tries to schedule tasks one-by-one in order. Any tasks that did not get scheduled remains in the system until the next scheduling time, and any unused unlocked budget remains available for future tasks. Users also specify a per-task timeout after which the task is evicted.

T is a parameter of the system that controls how many tasks get batched (and delayed) before getting scheduled. We evaluate its effect empirically in Fig. 9, and show that all algorithms we study are relatively insensitive to T.

4 Applicability

We now turn to two important considerations that arise when using DPK to schedule ML tasks in a training platform for continuous training on user data streams. We start by discussing the adaptivity in DP queries required by this use-case, and how it is supported by our design (§4.1). We then discuss the characteristics of workloads under which we expect DPK to yield improvements (§4.2).

4.1 Privacy Guarantees under Adaptive Workloads

An important DP consideration when building a system around DPK is that is that composition in the accounting mechanism we use needs to support adaptively chosen privacy budgets per task. Indeed in the online case, which is the main use-case when scheduling ML workloads on continuous data streams, both jobs submitted by developer and jobs scheduled by DPK can depend on past results. This is important, as DP composition requires special care when dealing with adaptive privacy budgets per task [52]. Fortunately privacy filters, a DP accounting mechanism enabling adaptive composition under a preset upper-bound on the privacy loss, are known for both basic accounting [52] and RDP accounting [16, 37].

To support a global (ε,δ)-DP guarantee under continuous data streams, we use the data block composition introduced by Sage [38, 41]: each data block is associated with a privacy filter, initiated with ε,δ for basic accounting, or $\varepsilon(\alpha)=\varepsilon-\frac{\log(1/\delta)}{\alpha-1}$ for RDP. The RDP initial value ensures that translating back to traditional DP with Eq. 1 ensures (ε,δ)-DP. A task is granted if, and only if, all filters grant the request (all blocks have enough budget left). This ensures the following property (proof in appendix):

Property 6. DPK enforces (ε,δ)-DP over adaptively chosen computations and privacy demands $\varepsilon_i(\alpha)$.

Proof. We provide a proof sketch following the structure used in [38] (Theorem 4.2) for basic composition. Each task has an (adaptive) RDP requirement for all blocks, with $\varepsilon(\alpha)=0$ for non-requested blocks. Each data block is associated with a privacy filter (Algorithm 1 in Lécuyer et al. [37]). A task runs if and only if all filters accept the task: applying Theorem 1 in Lécuyer et al. [37] ensures $\varepsilon(\alpha)=\varepsilon-\frac{\log(1/\delta)}{\alpha-1}$-RDP holds for each block. Applying Eq. 1 concludes the proof. □

4.2 When to Expect Improvements from DPK

It is worth reflecting on the characteristics of workloads under which DPK provides the most benefit compared to alternatives such as DPF. §3.1 gives examples of inefficient DPF operation with multiple blocks and alpha orders. However, DPF does not *always* behave inefficiently when invoked on multiple blocks or with multiple alpha orders. For example, if all the tasks in Fig. 1 uniformly demanded three blocks, then DPF would make the "optimal" choice. The same would happen if all the tasks in Fig. 3 had RDP curves that were all ordered in the same way across alphas, so that the ordering of highest demands is the same as the ordering of demands at the best alpha order. In such cases, DPK's "intelligence" – its appropriate treatment of the multiple blocks and focus on the best alpha – would not make a difference.

Instead, DPK should be expected to improve on DPF when the workload exhibits *heterogeneity* in one or both of these two dimensions: number of demanded blocks and best alphas. Indeed, the example in Fig. 1 exhibits high heterogeneity in demanded blocks, with Task 1 demanding three blocks while all the others demanding just one block. Similarly, the example in Fig. 3 exhibits heterogeneity in the best alpha for the different curves. This leads us to expect the greatest benefit from DPK for workloads with a high degree of heterogeneity in the number of demanded blocks and best alpha values of their RDP curves. §6.2 instills this intuition into concrete metrics that it uses to develop a microbenchmark that evaluates DPK and its baselines under a wide range of more or less heterogeneous workloads, confirming the effects discussed here.

We believe that heterogeneity of demands in both dimensions – number of blocks and best alphas – would be a realistic characteristic of real-world DP ML workloads. For example, a pipeline that computes some summary statistics over a dataset might run daily on just the latest block, while a large neural network may need to retrain on data from the past several blocks. This would result in heterogeneity in number of demanded blocks. Similarly, pipelines that compute simple statistics would likely employ a Laplace mechanism, while a neural network training would employ subsampled Gaussian. This would inevitably result in heterogeneity in best alphas, because, as shown in Fig. 2, different mechanisms exhibit very different RDP curves.

To summarize, DPK addresses heterogeneity in two orthogonal dimensions: multiple blocks and RDP alpha orders. DPK is therefore broadly applicable to: (1) systems that exhibit both of these dimensions (as would DP ML workloads in TFX-like systems, or static SQL databases with multiple partitions); (2) systems that operate on a single block (such as non-partitioned SQL databases) but perform RDP accounting; and (3) systems that operate on multiple blocks but perform other types of DP accounting, including traditional DP. For all these settings, DPK would provide a benefit when the workload exhibits heterogeneity.

5 Implementation

We implement DPK in two artifacts that we will open-source. The first is a **Kubernetes-based implementation** of DPK. We extend PrivateKube's extension to Kubernetes in multiple ways. We add support for batched scheduling (i.e. schedule tasks every T time units) and task weights. We implement DPK, and add support for solving the single block knapsack using Gurobi with the Go goop interface [21]. The Kubernetes-based implementation has 924 lines of Go.

The second artifact is a **simulator** that lets users easily specify and evaluate scheduling algorithms for the offline and online settings under different workloads. We use a discrete event simulator [55] to efficiently support arbitrarily fine time resolutions. Users use configuration files to define the workload and resource characteristics to parameterize scheduling for both online and offline cases. For example, they can define block and task arrival frequencies, the scheduling period and the block unlocking rate. The simulator also supports plug-

	Sec.	Workload	Setting	Prototype	Results
Q1	§6.2	microbenchmark	offline	simulator	Fig. 4
Q2	§6.2	microbenchmark	offline	simulator	Fig. 5
Q3	§6.3	Alibaba, Amazon	online	simulator	Fig. 6-7
Q4	§6.4	Alibaba	online	Kubernetes	Fig. 8

Tab. 1: **Workload and methodology of each evaluation question.**

ging different definitions of efficiency, and different block selection patterns for tasks (policies). Currently, the simulator supports two patterns: a random selection of blocks without replacement, and a selection of most recent blocks. The simulator has 6,718 lines of Python.

6 Evaluation

We seek to answer four evaluation questions:

Q1: On what types of workloads does DPK improve over DPF, and how close is DPK to Optimal?

Q2: How do the algorithms scale with increasing load?

Q3: Does DPK present an efficiency improvement for plausible workloads? How much does it trade fairness?

Q4: How does our implementation perform in a realistic setting?

These questions are best answered with distinct workloads and settings, summarized in Tab. 1. First, Q1 and Q2 are best addressed in an offline setting with a simple, tunable workload. To this end, we develop a microbenchmark consisting of multiple synthetic tasks with distinct RDP curves and a knob that controls the heterogeneity in demanded blocks and RDP curves (§6.2). Second, Q3 and Q4 require a more realistic, online setting and realistic workloads. In absence of a production trace of DP ML tasks, we develop a workload generator, called *Alibaba-DP*, based on Alibaba's 2022 ML cluster trace [58]. We map the Alibaba trace to a DP ML workload by mapping system metrics to privacy parameters (§6.3). While we cannot claim Alibaba-DP is realistic, it is the first *objectively-derived* DP task workload generator, and we believe it is a more plausible workload than those previously used in related works. We plan to release it publicly. Third, Q1-Q3 are algorithmic-level questions independent of implementation and hence we evaluate them in the simulator. However, Q4 requires an actual deployment on Kubernetes, so we dedicate the last part of this section to an evaluation on Kubernetes with the Alibaba-DP workload (§6.4).

6.1 Methodology

Baselines. The main baseline, common across all experiments, is *DPF*. We consider two other baselines: *Optimal*, which is the exact Gurobi-derived privacy knapsack solution for the offline setting, and *FCFS* (first-come-first-serve), which schedules tasks in an online setting based on their order of arrival. The former is relevant for offline experiments of small scale (few tasks/blocks), since it is not tractable for larger ones. The latter is relevant for online experiments only.

Metrics. *Global efficiency:* defined as either the number of allocated tasks or the sum of weighted allocated tasks. *Scheduler runtime:* measures how fast (in seconds), computationally, a scheduling algorithm is. *Scheduling delay:* measures how long tasks are blocked in the waiting queue, for example because of insufficient unlocked budget or because of the batching period T; it is measured in block inter-arrival periods (e.g., if blocks arrive daily, the unit is days). In real life, the total waiting time for a task will be the scheduling delay plus scheduler runtime; for our experiments, since the two are in different units, we never combine them. We expect in reality scheduler runtimes to be small compared to scheduling delays, for all the evaluated algorithms except for Optimal.

Machine. We use a server with 2 Intel Xeon CPUs E5-2640 v3 @ 2.60GHz (16 cores) and 110GiB RAM.

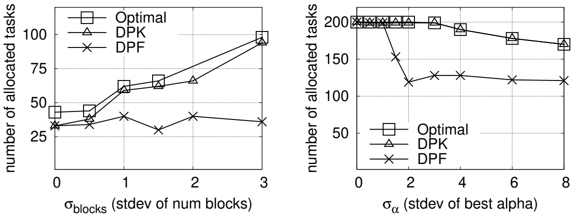
6.2 Offline Microbenchmark (Q1, Q2)

Microbenchmark. We design the microbenchmark to expose knobs that let us systematically explore a spectrum of workloads ranging from less to more heterogeneous in demanded blocks and RDP curve characteristics. The microbenchmark consists of 620 RDP curves corresponding to five realistic DP mechanisms often incorporated in DP ML workloads: {*Laplace, Subsampled Laplace, Gaussian, Subsampled Gaussian, composition of Laplace and Gaussian*}. We sample and parameterize these curves with the following methodology meant to expose two heterogeneity knobs:

Knob σ_{blocks}: To exercise heterogeneity in requested blocks, we sample the number of requested blocks from a discrete Gaussian with mean μ_{blocks} and standard deviation σ_{blocks}. The requested blocks are then chosen randomly from the available blocks. Increasing σ_{blocks} increases heterogeneity in demanded blocks.

Knob σ_α: To exercise heterogeneity in best alphas, we first normalize the demands (for a block with initial budget $(\varepsilon,\delta) = (10, 10^{-7})$) and enforce that there is at least one curve with best alpha α for each $\alpha \in \{3,4,5,6,8,16,32,64\}$. Second, we group tasks with identical best alphas to form "buckets". For each new task, we pick a best alpha following a truncated discrete Gaussian over the bucket's indexes, centered in the bucket corresponding to $\alpha = 5$ with standard deviation σ_α. Third, we sample one task uniformly at random from that bucket. After dropping some outliers (e.g. curves with $\varepsilon_{\min} < 0.05$), we rescale the curves to fit any desired value of the average and the standard deviation of $\varepsilon_{\min}$ for each best alpha, by shifting the curves up or down. This scaling lets us change the distribution in best alphas while controlling for the average size of the workload (in a real workload, the value of $\varepsilon_{\min}$ might be correlated with best alpha and other parameters). Increasing σ_α increases workload heterogeneity in best alphas.

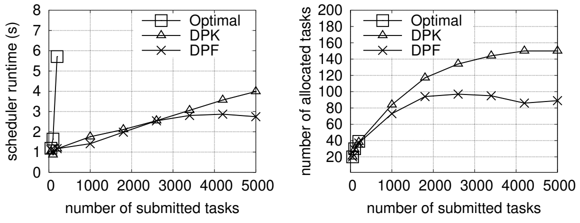
We explore each heterogeneity knob separately. First, we vary σ_{blocks} while keeping $\sigma_\alpha = 0$ (i.e. all the tasks have best alpha equal to 5) and $\mu_{blocks} = 10$. Second, we vary σ_α while keeping $\sigma_{blocks} = 0$, $\mu_{blocks} = 1$ (i.e. all the tasks request the same single block). In both cases, we keep $\varepsilon_{\min}$ constant for all tasks. We set $\varepsilon_{\min} = 0.1$ for the σ_{blocks} experiment (to



(a) Block heterogeneity

(b) Best alpha heterogeneity

Fig. 4: **(Q1) DPK under workloads with variable heterogeneity using our microbenchmark.** Shows the global efficiency of the algorithms (y axes) in the offline setting, as heterogeneity increases on the x axes in terms of: (a) variation in number of blocks requested and (b) variation in best alphas for the tasks' RDP curves. *Q1 Answer: DPK tracks Optimal closely and significantly outperforms DPF on workloads with high heterogeneity: 0–161% improvement for Fig. 4(a) and 0–67% for Fig. 4(b).*



(a) Scheduler runtime

(b) Allocated tasks

Fig. 5: **(Q2) Scalability under increasing load from the microbenchmark.** (a) Number of allocated tasks and (b) runtime of the scheduler, as function of offered load (x axes). *Q2 Answer: Optimal becomes intractable quickly while DPK and DPF remain practical even at high load.*

keep the number of tasks small enough to be tractable for Optimal) and $\varepsilon_{\min} = 0.005$ for the σ_α experiment (to have a large number of tasks with high diversity in $\varepsilon(\alpha)$).

Q1: On what types of workloads does DPK improve over DPF, and how close is DPK to Optimal? Fig. 4 compares the schedulers' global efficiency in the offline setting, as the heterogeneity of the workload increases in the two preceding dimensions: the number of requested blocks (Fig. 4(a)) and the best alphas of the tasks' RDP curves (Fig. 4(b)). Across the entire spectrum of heterogeneity, DPK closely tracks the optimal solution, staying within 23% of it. For workloads with low heterogeneity (up to 0.5 stdev in blocks and 1 stdev in best alphas), there is not much to optimize. DPF itself therefore performs close to Optimal and hence DPK does not provide significant improvement. As heterogeneity in either dimension increases, DPK starts to outperform DPF, presenting significant improvement in the number of allocated tasks for over 3 stdev in blocks and 2 stdev in best alphas: 161% and 67% improvement, respectively.

As all three schedulers try to schedule as many tasks as they can with a finite privacy budget, these 1.0–2.6× additional tasks that DPK is able to schedule are tasks that *DPF would never be able to schedule*, because the requested blocks' budget has been depleted for posterity.

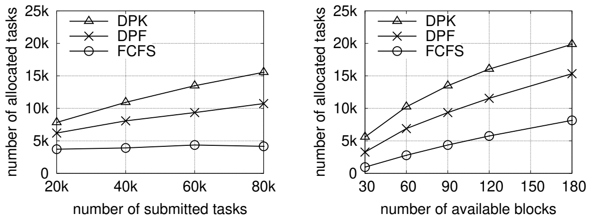
Q2: How do the algorithms scale with increasing load? Fig. 5(a) shows the runtime of our simulator on a single thread. We use a single thread for a fair comparison, but some schedulers can be parallelized (our Kubernetes implementation is indeed parallelized). We use the microbenchmark with heterogeneity knobs $\sigma_\alpha = 4, \sigma_{blocks} = 10, \mu_{blocks} = 1, \varepsilon_{\min} = 0.01$ and 7 available blocks. Optimal's line stops at $x = 200$ tasks because after that its execution never finishes. DPK takes slightly longer than DPF to run because it needs to solve multiple single-block knapsacks. Fig. 5(b) shows scheduler efficiency in number of allocated tasks as a function of the number of tasks in the system. DPF performs the worst, unable to efficiently schedule tasks across multiple blocks and varying alpha order demands. DPK matches Optimal (up to Optimal's 200 task limit) and schedules more tasks when it has a larger pool of tasks to choose from, since it can pick the most efficient tasks. Since the workload has a finite number of different tasks, as we increase the load, both schedulers reach a plateau where they allocate only one type of task.

6.3 Online Plausible Workload (Q3)

We now evaluate online scenarios where tasks and blocks arrive dynamically, and budget is unlocked over time. The simulator uses a virtual unit of time, where one block arrives each time unit. Tasks always request the m most recent blocks. For all the evaluated policies we run a batch scheduler on the available unlocked budget, every T blocks.

The Alibaba-DP Workload. We create a macrobenchmark based on Alibaba's GPU cluster trace [58]. The trace includes 1.1 million tasks submitted by 1,300 users over 3 months, and contains each task's resource demands and the resource allocation over time. We use these metrics as proxies for task DP budget demands, which do not exist in this trace.

We use machine type (CPU/GPU) as a proxy for DP mechanism type. We assume CPU-based tasks correspond to mechanisms used for statistics, analytics, or lightweight ML (e.g. XGBoost or decision trees [25, 39]), while GPU-based tasks correspond to deep learning mechanisms (DP-SGD or DP-FTRL [1, 34]). We map each CPU-based task to one of the {*Laplace, Gaussian, Subsampled Laplace*} curves and each GPU-based task to one of the {*composition of Subsampled Gaussians, composition of Gaussians*} curves, at random. We use memory usage as a proxy for privacy usage by setting traditional DP ε as an affine transformation of memory usage (in GB hours). We don't claim that memory will be correlated with privacy in a realistic DP workload, but that the privacy budget might follow a similar distribution (e.g. a power law with many tasks having small requests and a long tail of tasks with large requests). We compute the number of blocks required by each task as an affine function of the bytes read through the network. Unlike the privacy budget proxy, we expect this proxy to have at least some degree of realism when data is stored remotely: tasks that don't communicate much over the network are probably not using large portions of the dataset. Finally, all tasks request the most recent blocks that



(a) Allocated tasks as function of submitted tasks

(b) Allocated tasks as function of number of available blocks

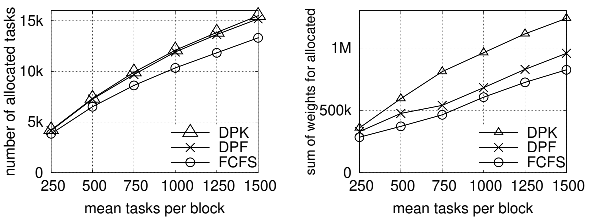
Fig. 6: **(Q3) Efficiency evaluation on the online Alibaba-DP workload.** Number of allocated tasks as a function of (a) offered load for 90 blocks and (b) available blocks for 60k tasks. *Q3 Answer: Alibaba-DP exhibits sufficient heterogeneity for DPK to present a significant improvement (1.3–1.7×) over DPF.*

arrived in the system and are assigned a weight of 1. We truncate the workload by sampling one month of the total trace and cutting off tasks that request more than 100 blocks or whose smallest normalized RDP ε is not in $[0.001, 1]$. The resulting workload, called *Alibaba-DP*, is an objectively derived version of the Alibaba trace. We use it to evaluate DPK under a more complex workload than our synthetic microbenchmark or PrivateKube's also synthetic workload.

Q3: Does DPK present an efficiency improvement for plausible workloads? How much does it trade fairness? Fig. 6(a) shows the number of allocated tasks as a function of the number of submitted ones from the Alibaba-DP workload. The results show that as the number of submitted tasks increases, both DPF and DPK can allocate more tasks, because they have a larger pool of low-demand submitted tasks to choose from. This is not the case with FCFS, which does not prioritize low-demand tasks. DPK allocates 22–43% more tasks than DPF, since it packs the tasks more efficiently. Similarly, Fig. 6(b) shows the number of allocated tasks as a function of the number of available blocks. As expected, all algorithms can schedule more tasks when they have more available budget. DPK consistently outperforms DPF, scheduling 30–71% more tasks. Across all the configurations evaluated in Fig. 6(a) and 6(b), DPK outperforms DPF by 1.3–1.7×. The results confirm that Alibaba-DP, a workload derived objectively from a real trace, exhibits sufficient heterogeneity for DPK to show significant benefit.

The appendix (Fig. 9) includes an evaluation of the number of allocated tasks as a function of T. We find that T has very little impact on the algorithms' performance, and can be set to a low value to minimize scheduling delay.

Efficiency–Fairness Trade-off. While DPK schedules significantly more tasks than DPF on the Alibaba workload, this increased efficiency comes at the cost of fairness, when we use DPF's definition of fairness. To demonstrate this, we run the Alibaba workload with 90 blocks and 60k tasks, and set the DPF "fair share" of tasks to be $\frac{1}{50}$. This means that DPF will always prioritize tasks that request $\frac{1}{50}$ or less of the epsilon-normalized global budget. In the Alibaba trace, using this definition, 41% of tasks would qualify as demanding less or equal budget than their fair share. With DPK, 60% of the allocated tasks are fair-share tasks; with DPF 90% are. However, DPK can allocate 45% more tasks than DPF. As expected, this shows that optimizing for efficiency comes at the expense of fairness. In the case of privacy scheduling, however, due to the finite nature of the privacy budget, DPF's fairness guarantees are limited only to the first N fair share tasks (in the experiment, $N = 50$); the guarantees do *not* hold for later-arriving tasks. This makes the overall notion of fairness as defined by DPF somewhat arbitrary and underscores the merit of efficiency-oriented algorithms.



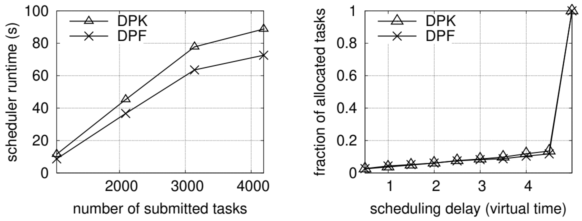
(a) Original workload

(b) Workload with task weights

Fig. 7: **Evaluation on Amazon Reviews workload from [40].** (a) This workload, which is synthetic and very simple, exhibits limited heterogeneity, so there is no room for DPK to improve over DPF. (b) Adding randomly selected weights to the tasks creates sufficient heterogeneity for DPK to mark an improvement. Global efficiency is measured as the sum of weights of allocated tasks (y axis).

Another workload: Amazon Reviews [40]. We also evaluate on the macrobenchmark workload from the PrivateKube paper [40], which consists of several DP models trained on the Amazon Reviews dataset [47]. Unlike our Alibaba-DP, which is rooted in a real ML workload trace, this workload is completely synthetic and very small, and as a result, its characteristics may be very different from real workloads. Yet, for completeness, we evaluate it here, too. The workload consists two categories of tasks: 24 tasks to train neural networks with a composition of subsampled Gaussians, and 18 tasks to compute summary statistics with Laplace mechanisms. Unlike for our Alibaba-DP workload, task arrival needs to be configured for this workload; tasks arrive with a Poisson process and request the latest blocks. The Amazon Reviews workload has low heterogeneity both in terms of block and the best-alpha variance. Although tasks request up to 50 blocks, 95% of the tasks in this workload request 5 blocks or fewer, and 63% of the tasks request only 1 block. Moreover, tasks have only 2 possible best alphas (4 or 5), with 81% of the tasks with a best alpha of 5. Hence, per our Q1 results in §6.2, we expect DPF to already perform well and leave no room for improvement for DPK. Fig. 7(a) confirms this: all schedulers perform largely the same on this workload.

Next, without modifying the privacy budget or the blocks they request, we configure different weights for submitted tasks, corresponding to different profits the company might get if a task gets to run. We assume that large tasks (neural networks) are more important than small tasks. Then, we pick



(a) Scheduler runtime

(b) Scheduling delay CDF

Fig. 8: **(Q4) Evaluation on Kubernetes with Alibaba-DP.** (a) Scheduler runtime as function of submitted tasks in offline experiment ($T = 25$), and (b) CDF of scheduling delay (we exclude scheduler runtime) for allocated tasks in online experiment ($T = 5$). *Q4 Answer: DPK has only a modestly higher runtime than DPF, because system-related overheads dominate runtime. In an online setting (b), scheduling delays are almost identical across schedulers.*

Scheduler	Number of allocated tasks
DPK	1269
DPF	1100

Tab. 2: **Efficiency on Kubernetes prototype with Alibaba-DP.**

an arbitrary grid of weights while still allowing some small tasks to be more profitable than some large tasks. Weights are chosen uniformly at random from $\{10, 50, 100, 500\}$ for large tasks and $\{1, 5, 10, 50\}$ for small tasks. This change implicitly scales the number of requested blocks and increases heterogeneity. In terms of global efficiency, a task with weight k demanding m blocks is roughly similar to a task with weight 1 demanding m/k blocks. Instead of having most tasks request 1 block, tasks now demand a higher-variance weighted number of blocks (the variation coefficient is 1.9 instead of 1.3). Fig. 7(b) shows the global efficiency, measured as the sum of weights of allocated tasks, as a function of the number of submitted tasks. DPK now outperforms DPF by 9–50%.

6.4 Kubernetes Implementation Evaluation (Q4)

Q4: How does our implementation perform in a realistic setting? We evaluate the Alibaba-DP workload on our Kubernetes system. *Scheduler runtime:* We first estimate the scheduler's overhead by emulating an offline scenario, where all the tasks and blocks are available. To do so, we use a large $T = 25$. For this experiment, we generate a total of 4,190 tasks by sampling 2 days of the Alibaba cluster trace. The experiment shows the runtime as a function of the number of submitted tasks. It uses 10 offline and 20 online blocks. Fig. 8(a) shows the total time spent in the scheduling procedure, which includes Kubernetes-related overheads (e.g. inter-process communication and synchronization). As noted in §5(a), DPK has a higher overhead since it solves knapsack subproblems. DPK has a higher runtime overhead than DPF since it has to recompute the efficiency of each task when the global state changes after a scheduling cycle, while DPF computes the dominant share of each task only once. Nevertheless, the overhead is modest, because: (a) the Kubernetes overheads dominate, and (b) the DPK (and DPF) algorithms are parallelized. In addition, since DP tasks are often long-running (e.g. distributed training of deep neural networks), the scheduling delay of DPK in many cases is insignificant compared to the total task completion time.

Scheduling delays and efficiency: We run an experiment to measure the scheduling delays (Fig. 8(b)) and efficiency (Table 2) in an online scenario on Kubernetes. We use the same workload and number of blocks as in Fig. 8(a), with $T = 5$. As before, DPK is more efficient than DPF. Scheduling delay, measured in virtual time, excluding scheduler runtime, shows no significant difference between the two policies.

7 Related Work

We have already covered the details of the most closely related works: DPF and related systems for privacy scheduling [38, 40] (background in §2.3, efficiency limitations in §3.1 and §3.2, and experimental evaluation in §6). To summarize, we adopt their threat and system models, but instead of focusing on fairness, we focus on efficiency because we believe that the biggest pressure in globally-DP ML systems will ultimately be how to fit as many models as possible under a meaningful privacy guarantee. We discuss other related works next.

Bin packing for data-intensive tasks. Multidimensional knapsack and bin packing are classic NP-hard problems [2, 36, 60]. In recent years, several heuristics for these problems have been proposed to increase resource utilization in big data and ML clusters [22–24, 26]. Some of these heuristics assign a weight to each dimension and reduce to a scalar problem with a dot product [35, 49]. We show that the Rényi formulation of differential privacy generates a new variation of the multidimensional knapsack problem, making standard approximations and heuristics unsuitable.

Scheduling trade-offs. Fairness and performance is a classic tradeoff in scheduling even in single-resource scenarios. Shortest-remaining-time-first (SRTF) is optimal for minimizing the average completion time, but it can be unfair to long-running tasks and cause starvation. Recent works have shown a similar fairness and efficiency tradeoff in the multi-resource setting [23]. Although max-min fairness can provide both fairness and efficiency for a single resource, its extension to multi-resource fairness [17] can have arbitrarily low efficiency in the worst case [8]. In this paper, we highlight the fairness-efficiency tradeoff when allocating privacy blocks among multiple tasks with RDP.

Differential privacy. The literature on *DP algorithms* is extensive, including theory for most popular ML algorithms (e.g. SGD [1, 62], federated learning [42]) and statistics (e.g. contingency tables [4], histograms [61]), and their open source implementations [15, 18, 19, 31, 48]. These lower-level algorithms run as tasks in our workloads. Some algorithms focus on workloads [29], including on a data streams [9], but they remain limited to linear queries. Some *DP systems* also exist, but most do not handle ML workloads, instead providing DP SQL-like [43, 51, 59] and MapReduce interfaces [53], or support for summary statistics [45]. Sage [38] and PrivateKube [40], previously discussed, handle ML workloads.

8 Conclusions

This paper for the first time explores how data privacy should be scheduled efficiently as a computing resource. It formulates the scheduling problem as a new type of multidimensional knapsack optimization, and proposes and evaluates an approximate algorithm, DPK, that is able to schedule significantly more tasks than the state-of-the-art. By taking the first step of building an efficient scheduler for DP, we believe this work builds a foundation for tackling several important open challenges for managing access to DP in real-world settings, such as supporting tasks with different utility functions, investigating job-level scheduling, and better scheduling of traditional computing resources alongside privacy blocks.

References

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

[2] Y. Azar, I. R. Cohen, S. Kamara, and B. Shepherd. Tight bounds for online vector bin packing. In *Proceedings of the forty-fifth annual ACM symposium on Theory of Computing*, pages 961–970, 2013.

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

[4] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. 2007.

[5] S. Berghel, P. Bohannon, D. Desfontaines, C. Estes, S. Haney, L. Hartman, M. Hay, A. Machanavajjhala, T. Magerlein, G. Miklau, A. Pai, W. Sexton, and R. Shrestha. Tumult Analytics: a robust, easy-to-use, scalable, and expressive framework for differential privacy. *arXiv*, Dec. 2022.

[6] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In N. Heninger and P. Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 267–284. USENIX Association, 2019.

[7] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel. Extracting training data from large language models. In M. Bailey and R. Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2633–2650. USENIX Association, 2021.

[8] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica. HUG: Multi-resource fairness for correlated and elastic demands. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 407–424, Santa Clara, CA, Mar. 2016. USENIX Association.

[9] R. Cummings, S. Krehbiel, K. A. Lai, and U. Tantipongpipat. Differential privacy for growing databases. 2018.

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

[11] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Proc. of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, 2006.

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

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

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

[15] Facebook. Opacus. https://opacus.ai/. Accessed: 2020-11-10.

[16] V. Feldman and T. Zrnic. Individual privacy accounting via a renyi filter. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[17] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In D. G. Andersen and S. Ratnasamy, editors, *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*. USENIX Association, 2011.

[18] Google. Differential Privacy. https://github.com/google/differential-privacy/. Accessed: 2020-11-10.

[19] Google. TensorFlow Privacy. https://github.com/tensorflow/privacy. Accessed: 2020-11-10.

[20] Google Differential Privacy. https://github.com/google/differential-privacy/tree/main/python/dp_accounting, 2022.

[21] Goop generalized mixed integer optimization in Go. Goop homepage. https://github.com/mit-drl/goop/, 2021.

[22] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 455–466, New York, NY, USA, 2014. Association for Computing Machinery.

[23] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan. Altruistic scheduling in multi-resource clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 65–80, Savannah, GA, Nov. 2016. USENIX Association.

[24] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni. Graphene: Packing and dependency-aware scheduling for data-parallel clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 81–97, USA, 2016. USENIX Association.

[25] N. Grislain and J. Gonzalvez. Dp-xgboost: Private machine learning at scale. *CoRR*, abs/2110.12770, 2021.

[26] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. H. Liu, and C. Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *USENIX NSDI*, pages 485–500, 2019.

[27] Gurobi Optimization. Gurobi Optimization homepage. `www.gurobi.com/`, 2021.

[28] A. Gutman and N. Nisan. Fair allocation without trade. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, page 719–728, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[29] M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *Symposium on Foundations of Computer Science*, 2010.

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

[31] IBM. Diffprivlib. `https://github.com/IBM/differential-privacy-library`. Accessed: 2020-12-7.

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

[33] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multiresource allocation: Fairness–efficiency tradeoffs in a unifying framework. *IEEE/ACM Transactions on Networking*, 21(6):1785–1798, 2013.

[34] P. Kairouz, B. McMahan, S. Song, O. Thakkar, A. Thakurta, and Z. Xu. Practical and private (deep) learning without sampling or shuffling. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 5213–5225. PMLR, 2021.

[35] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[36] L. T. Kou and G. Markowsky. Multidimensional bin packing algorithms. *IBM Journal of Research and development*, 21(5):443–448, 1977.

[37] M. Lécuyer. Practical Privacy Filters and Odometers with Rényi Differential Privacy and Applications to Differentially Private Deep Learning. In *arXiv, v2*, 2021.

[38] M. Lécuyer, R. Spahn, K. Vodrahalli, R. Geambasu, and D. Hsu. Privacy Accounting and Quality Control in the Sage Differentially Private ML Platform. In *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[39] Q. Li, Z. Wu, Z. Wen, and B. He. Privacy-preserving gradient boosting decision trees. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 784–791. AAAI Press, 2020.

[40] T. Luo, M. Pan, P. Tholoniat, A. Cidon, R. Geambasu, and M. Lécuyer. Privacy budget scheduling. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 55–74. USENIX Association, July 2021.

[41] T. Luo, M. Pan, P. Tholoniat, A. Cidon, R. Geambasu, and M. Lécuyer. Privacy Resource Scheduling (extended version). `https://github.com/columbia/privatekube`, 2021.

[42] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. 2018.

[43] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. 2009.

[44] I. Mironov. Rényi Differential Privacy. In *Computer Security Foundations Symposium (CSF)*, 2017.

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

[46] J. Murtagh and S. Vadhan. The Complexity of Computing the Optimal Composition of Differential Privacy. In *Theory of Cryptography*, pages 157–175. Springer, Berlin, Germany, Dec. 2015.

[47] J. Ni, J. Li, and J. McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).*, pages 188–197, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. `https://nijianmo.github.io/amazon/index.html`.

[48] OpenDP. `https://smartnoise.org/`. Accessed: 2020-11-10.

[49] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder. Heuristics for vector bin packing. Technical report, 2011.

[50] D. C. Parkes, A. D. Procaccia, and N. Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. *ACM Transactions on Economics and Computation (TEAC)*, 3(1):1–22, 2015.

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

[52] R. M. Rogers, A. Roth, J. Ullman, and S. Vadhan. Privacy odometers and filters: Pay-as-you-go composition. 2016.

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

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

[55] Simpy. Discrete event simulation for Python. `https://simpy.readthedocs.io/en/latest/index.html`, 2020.

[56] TensorFlow Extended Guide. `https://www.tensorflow.org/tfx/guide/examplegen`, 2022.

[57] S. Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*. 2017.

[58] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, Renton, WA, Apr. 2022. USENIX Association.

[59] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson. Differentially private sql with bounded user contribution. *Proceedings on Privacy Enhancing Technologies*, 2020(2):230–250, 2020.

[60] G. J. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.

[61] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. 2012.

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

[63] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau. Ektelo: A framework for defining differentially-private computations. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 115–130, New York, NY, USA, 2018. Association for Computing Machinery.

A Formal Proofs

Property 1. The decision problem for the privacy knapsack problem is NP-hard.

Proof. Consider the decision problem corresponding to privacy knapsack: we are given an instance (demands, weights and capacities), and given a sum of weights $t \in \mathbb{R}$ we have to decide whether t is achievable by an allocation. We assume the number of RDP orders $|A|$ is fixed.

We can build a polynomial time reduction from the knapsack problem (KP) to the privacy knapsack problem (PK).

Consider an instance (d, w, c) of KP with demands $d = (d_1, \dots, d_n)$ and weights $w = (w_1, \dots, w_n)$. We define an instance $f(d', w', c')$ of PK with 1 block ($j = 1$) and n tasks such that for each task $i \in [n]$ and $\alpha \in A$: $d'_{ij\alpha} = d_i$ (the demands are identical for all the α orders), $c'_{j\alpha} = c$ (idem), and $w'_i = w_i$.

Thanks to this mapping, we have KP $\leq_p$ PK (i.e. if we can solve PK, then we can solve KP with polynomial overhead), because f is poly-time computable and $(d, w, c) \in \text{KP} \iff f(d', w', c') \in \text{RK}$. Indeed,

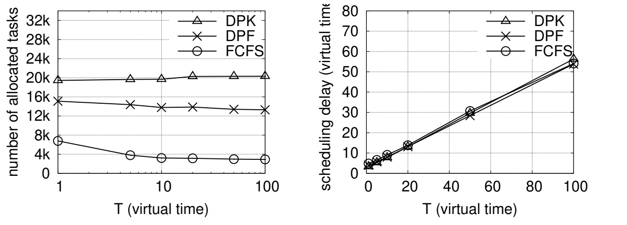
- If we have a KP allocation with sum of allocated weights t, we can reuse the exact same allocation in $f(d', w', c')$. The privacy knapsack capacity constraint will be satisfied for $\alpha = 1$ (for example).
- On the other hand, given a PK allocation $f(d', w', c')$ that achieves total weight $t > 0$, we can build an allocation in KP that achieves the same weight by selecting any α.

We know that KP is NP-hard, thus PK is NP-hard. □

Property 2. In the single-block case, there is a fully polynomial time approximation scheme (FPTAS) for the privacy knapsack problem.

Proof. First, we know that the 1-dimensional Knapsack Problem (KP) is in FPTAS. For example, for $1 > \varepsilon > 0$ and $n \in \mathbb{N}$, prior work [35] gives an $(1-\varepsilon)$-approximation algorithm $\mathcal{K}$ for KP with runtime $O(n^3/\varepsilon)$ on instances of size n. Consider a single block $j = 1$, n tasks and $|A|$ RDP orders. We can approximate the maximum profit for PK by running $|A|$ instances of $\mathcal{K}$:

1. For order α, compute the (approximate) solution $\hat{w}^{\max}_{\alpha}$ for $\max_x \sum_i w_i x_i$ subject to $\sum_i d_{ij\alpha} x_i \leq c_{j\alpha}$ using $\mathcal{K}$.



(a) Allocated tasks (b) Scheduling delay

Fig. 9: **Impact of batching parameter T on global efficiency (a) and scheduling delay (b).** (a) Number of allocated tasks, and (b) scheduling delay (in virtual time), both as a function of the batching parameter T (in virtual time). *In terms of allocated tasks, all schedulers are relatively insensitive to the batching parameter.*

2. Then, output the maximum profit $\hat{w}^{\max} := \max_{\alpha \in A} \hat{w}^{\max}_{\alpha}$.

Since A is fixed, this algorithm also runs in $O(n^3/\varepsilon)$ time, and it is an $(1-\varepsilon)$-approximation algorithm for PK. Indeed, there exists an $\alpha \in A$ such that $w^{\max}_{\alpha} \geq w^{\max}$ where $w^{\max} := \max_x \sum_i w_i x_i$ subject to $\exists \alpha' \in A, \sum_i d_{ij\alpha'} x_i \leq c_{j\alpha'}$. Finally, the output satisfies $\hat{w}^{\max} \geq \hat{w}^{\max}_{\alpha} \geq (1-\varepsilon) w^{\max}_{\alpha} \geq (1-\varepsilon) w^{\max}$. □

Property 3. For $m \geq 2$ blocks, there is no FPTAS for the privacy knapsack problem unless P=NP.

Proof. We know that there is no FPTAS for the multidimensional knapsack problem (d-KP) when $d \geq 2$, unless P=NP [35]. We can reuse the reduction from Prop. 2 to solve d-KP from PK with $m = d$ blocks with polynomial overhead. □

B Additional Experiments

Fig. 9 shows the number of allocated tasks as a function of T. With a high T, the online setting converges to the offline setting. DPK and DPF perform more or less the same as a function of T, while FCFS performs worse. This is because with a high T, more budget will be unlocked to schedule large tasks that arrived early, which would otherwise not get scheduled if their budget was not yet unlocked. DPK consistently outperforms DPF by 28–52% in this experiment. Therefore, we can conclude that T can be safely set to a relatively low value (to minimize scheduling delay).