Journal of Privacy and Confidentiality
Vol. 15 (3) 2025

TPDP 2024

Submitted April 2025
Published December 2025

BRIDGING THE PRIVACY ACCOUNTING GAP IN DP-SGD *

LYNN CHUA, BADIH GHAZI, CHARLIE HARRISON, ETHAN LEEMAN, PRITISH KAMATH, RAVI KUMAR, PASIN MANURANGSI, AMER SINHA, AND CHIYUAN ZHANG

Google Research
e-mail address: chualynn@google.com

e-mail address: badihghazi@gmail.com

e-mail address: csharrison@google.com

e-mail address: ethanleeman@google.com

e-mail address: pritish@alum.mit.edu

e-mail address: ravi.k53@gmail.com

e-mail address: pasin@google.com

e-mail address: amersinha@google.com

e-mail address: chiyuan@google.com

Abstract. Differentially Private Stochastic Gradient Descent (DP-SGD) is one of the most widely used algorithms for private machine learning. Due to its efficiency, most practical implementations of DP-SGD shuffle the training examples and divide them into fixed-size mini-batches during training. However, the privacy accounting typically assumes that Poisson subsampling was used, wherein each example is included in each mini-batch independently with some probability. Our first contribution is to show that there can be a substantial gap between these two versions of DP-SGD; specifically, the privacy accounting implies much stronger privacy guarantees than the implementations actually provide. As our second contribution, we propose two approaches to address this gap: (i) an implementation of Poisson subsampling using the Map-Reduce framework that can scale to large datasets that do not fit in memory and (ii) a novel *Balls-and-Bins* sampling that achieves the "best of both" sampling approaches. Namely, its implementation is similar to shuffling, and it leads to similar utility for DP-SGD training with similar-or-better privacy compared to Poisson subsampling in practical regimes of parameters.

Introduction

Training differentiable models, e.g., neural networks, with noisy gradients via first-order methods such as stochastic gradient descent (SGD), has become a common approach for making training pipelines satisfy differential privacy (DP). Since its introduction by Abadi

Key words and phrases: Differential Privacy, DP-SGD, Shuffling, Poisson Subsampling, Balls-and-Bins sampling.

* This paper combines the works [Chua et al., 2024a,b, 2025], which previously appeared at ICML 2024, NeurIPS 2024, and AISTATS 2025, respectively.

JPrivacy Confidentiality www.journalprivacyconfidentiality.org
DOI:10.29012/jpc.998

et al. [2016], this formulation of DP-SGD has been the basis of open source implementations in Google [2025], PyTorch Opacus [Yousefpour et al., 2021], and JAX Privacy [Balle et al., 2022]. DP-SGD has been widely applied across various domains [e.g., De et al., 2022, Dockhorn et al., 2023, Anil et al., 2022, He et al., 2023, Igamberdiev et al., 2024, Tang et al., 2025].

DP-SGD processes the training data in a sequence of steps, where at each step, a noisy estimate of the average gradient over a mini-batch (henceforth referred to as "batch" for brevity) is computed and used to perform a first-order update over the differentiable model; a formal description is provided in Algorithm 1. In summary, the noisy (average) gradient is obtained by *clipping* the gradient g for each example in the batch to have its norm at most C (a preset bound), namely $[g]_C := g \cdot \min\{1, C/\|g\|_2\}$, computing the sum over the batch, and then adding independent zero-mean noise drawn from the Gaussian distribution of scale σC to each coordinate of the gradient sum. This could then be normalized to obtain a noisy average gradient.[1] The privacy guarantee of the mechanism depends on the following: the noise scale σ, the number of examples in the training dataset n, the (expected) size of batches b, the number of training steps T, and the *batch generation process*.

Most deep learning systems in recent years process batches of fixed-size by sequentially iterating over the dataset, perhaps after applying a global or some other form of *shuffling* to the dataset. The privacy analysis for such a mechanism however has been technically challenging due to the correlated nature of the batches. To simplify the privacy analysis, Abadi et al. [2016] considered *Poisson subsampling*, wherein each batch is sampled independently by including each example independently with a fixed probability. However, Poisson subsampling is rarely implemented and instead it has become common practice to use some form of shuffling in model training, but to report privacy parameters as if Poisson subsampling was used [Ponomareva et al., 2023, §4.3]; a notable exception is the PyTorch Opacus library [Yousefpour et al., 2021] that supports Poisson subsampling for DP-SGD and provides privacy accounting methods for it.

Adaptive Batch Linear Queries. Formally, the privacy analysis of DP-SGD, especially in the case of non-convex models such as deep neural networks, is typically performed by viewing it as a post-processing of a mechanism performing *adaptive batch linear queries*

[1] When the batch size is a random variable, the "normalization" has to be done by scaling with a fixed value, e.g., the expected batch size b, and not the realized batch size.

Algorithm 1: DP-SGD$_{\mathcal{G}}$ [Abadi et al., 2016].

Parameters: Differentiable loss $\ell : \mathbb{R}^d \times \mathcal{X} \to \mathbb{R}^d$, initial state w_0, clipping norm C, noise scale σ, batch generator $\mathcal{G}_{b,T}$ that samples T batches, with (expected) batch size b.

Input: Dataset $\boldsymbol{x} = (x_1, \ldots, x_n)$.

Output: Final model state $\boldsymbol{w}_T \in \mathbb{R}^d$.

$(S_1, \ldots, S_T) \leftarrow \mathcal{G}_{b,T}(n)$

for $t = 1, \ldots, T$ **do**

  $g_t \leftarrow \frac{1}{b}\left(\sum_{x \in S_t} [\nabla_{\boldsymbol{w}} \ell(\boldsymbol{w}; x)]_C + \mathcal{N}(0, \sigma^2 C^2 I_d)\right)$

  $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} - \eta_t g_t$ ▷ or any other optimizer step; η_t is the learning rate

end

return $\boldsymbol{w}_T$

Algorithm 2: $\mathsf{ABLQ}_{\mathcal{G}}$: Adaptive Batch Linear Queries.

Parameters: Batch generator $\mathcal{G}$, noise scale σ, and (adaptive) query method $\mathcal{A} : (\mathbb{R}^d)^* \times \mathcal{X} \to \mathbb{B}^d$.
Input: Dataset $\boldsymbol{x} = (x_1, \ldots, x_n)$.
Output: Query estimates $g_1, \ldots, g_T \in \mathbb{R}^d$
$(S_1, \ldots, S_T) \leftarrow \mathcal{G}(n)$
for $t = 1, \ldots, T$ **do**
  $\psi_t(\cdot) := \mathcal{A}(g_1, \ldots, g_{t-1}; \cdot)$
  $g_t \leftarrow \sum_{i \in S_t} \psi_t(x_i) + e_t$ for $e_t \sim \mathcal{N}(0, \sigma^2 I_d)$
end
return $(g_1, \ldots, g_T)$

Algorithm 3: $\mathcal{D}_{b,T}$: Deterministic Batch Generator.

Parameters: Batch size b, number of batches T.
Input: Number of datapoints $n = b \cdot T$.
Output: Sequence of disjoint batches $S_1, \ldots, S_T \subseteq [n]$.
for $t = 0, \ldots, T-1$ **do**
  $S_{t+1} \leftarrow \{tb + 1, \ldots, tb + b\}$
end
return $S_1, \ldots, S_T$

($\mathsf{ABLQ}_{\mathcal{G}}$, as formalized in Algorithm 2) that releases estimates of a sequence of adaptively chosen linear queries as produced by an *adaptive query method* $\mathcal{A}$, on the batches obtained using a *batch generator* $\mathcal{G}$ (we use subscript $\mathcal{G}$ to emphasize the role of the batch generator). A batch generator $\mathcal{G}$ can be any algorithm that generates a sequence $S_1, \ldots, S_T \subseteq [n]$ of batches. We use $\mathcal{G}_{b,T}$ to emphasize the number of batches T and the (expected) batch size b, but often omit the subscript when it is clear from context.

Given a dataset of n examples, $\mathsf{ABLQ}_{\mathcal{G}}$ processes the batches generated by $\mathcal{G}$ in a sequential order to produce a sequence $(g_1, \ldots, g_T)$. Here, each $g_t \in \mathbb{R}^d$ is the sum of $\psi_t(x)$ over the batch S_t with added zero-mean Gaussian noise of scale σ to all coordinates, where the query $\psi_t : \mathcal{X} \to \mathbb{B}^d$ (for $\mathbb{B}^d := \{v \in \mathbb{R}^d : \|v\|_2 \leq 1\}$) is produced by the adaptive query method $\mathcal{A}$, based on the previous responses $g_1, \ldots, g_{t-1}$. DP-SGD with batch generator $\mathcal{G}$ (denoted $\mathsf{DP\text{-}SGD}_{\mathcal{G}}$) can be obtained as a post-processing of the output of $\mathsf{ABLQ}_{\mathcal{G}}$, with the adaptive query method that maps examples to the (scaled) clipped gradient at the current iterate, namely $\psi_t(x) := [\nabla_{\boldsymbol{w}} \ell(\boldsymbol{w}_{t-1}, x)]_C / C$ where $\boldsymbol{w}_{t-1} := -\frac{C}{b} \sum_{\ell=1}^{t-1} \eta_\ell g_\ell$.

As mentioned above, a canonical way to generate the batches is to go through the dataset in a fixed deterministic order, and divide the data into batches of a fixed size (Algorithm 3, denoted as $\mathcal{D}$). Another common option is to first randomly permute the entire dataset before dividing it into batches of a fixed size (Algorithm 4, denoted as $\mathcal{S}$); this option provides *amplification by shuffling*, namely, that the privacy guarantees are better compared to fixed deterministic ordering. However, obtaining such amplification bounds is non-trivial, and while some bounds have been recently established for such privacy amplification [Erlingsson et al., 2019, Feldman et al., 2022, Feldman et al.], they tend to be loose in our setting and only kick in when the basic mechanism is already sufficiently private.

Algorithm 4: $\mathcal{S}_{b,T}$: Shuffle Batch Generator.

Parameters: Batch size b, number of batches T.
Input: Number of datapoints $n = b \cdot T$.
Output: Sequence of disjoint batches $S_1, \ldots, S_T \subseteq [n]$.
Sample a random permutation π over $[n]$.
for $t = 0, \ldots, T-1$ **do**
  $S_{t+1} \leftarrow \{\pi(tb+1), \ldots, \pi(tb+b)\}$
end
return $S_1, \ldots, S_T$

Algorithm 5: $\mathcal{P}_{b,T}$: Poisson Batch Generator.

Parameters: Expected batch size b, number of batches T.
Input: Number of datapoints n.
Output: Sequence of batches $S_1, \ldots, S_T \subseteq [n]$.
for $t = 1, \ldots, T$ **do**
  $S_t \leftarrow \emptyset$
  for $i = 1, \ldots, n$ **do**
    $S_t \leftarrow \begin{cases} S_t \cup \{i\} & \text{with probability } b/n \\ S_t & \text{with probability } 1 - b/n \end{cases}$
  end
end
return $S_1, \ldots, S_T$

Instead, the approach followed by Abadi et al. [2016], and henceforth used commonly in reporting privacy parameters for DP-SGD, is to assume that each batch is sampled i.i.d. by including each record with a certain probability, referred to as *Poisson subsampling* (Algorithm 5, denoted as $\mathcal{P}$); the expected batch size b in this case need not be an integer, but for sake of uniformity, we assume that b is an integer. The advantage of this approach is that its privacy analysis is easier to carry out since the $\mathsf{ABLQ}_{\mathcal{P}}$ mechanism can be viewed as a composition of T independent sub-mechanisms. This enables privacy accounting methods such as Rényi-DP [Mironov, 2017] as well as numerically tight accounting methods using privacy loss distributions (elaborated on later in Section 2).

This has been the case, even when the algorithm being implemented in fact uses some form of shuffling-based batch generator. To quote Abadi et al. [2016] (with our emphasis),

> "In practice, for efficiency, the construction of batches and lots is done by randomly permuting the examples and then *partitioning* them into groups of the appropriate sizes. *For ease of analysis*, however, we assume that *each lot is formed by independently picking each example with probability* $q = L/N$, where N is the size of the input dataset."

Most implementations of DP-SGD mentioned earlier also use some form of shuffling, with the rare exception of PyTorch Opacus [Yousefpour et al., 2021] that has the option of Poisson subsampling to be consistent with the privacy analysis. But this approach does not scale to large datasets as random access for datasets that do not fit in memory is generally inefficient. Moreover, variable batch sizes are inconvenient to handle in deep

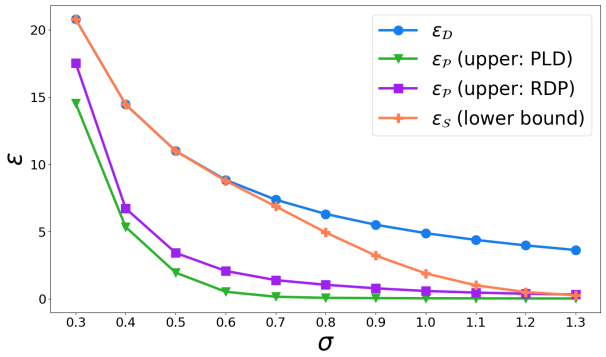

Figure 1: Privacy parameter ε for different noise parameters σ, for fixed $\delta = 10^{-6}$ and number of steps $T = 10\,000$. $\varepsilon_{\mathcal{D}}$: for deterministic batching, $\varepsilon_{\mathcal{P}}$: upper bounds when using Poisson subsampling (computed using different accountants), and $\varepsilon_{\mathcal{S}}$: a lower bound when using shuffling. We observe that shuffling does not provide much amplification for small values of σ, incurring significantly higher privacy cost compared to Poisson subsampling.

learning systems (see Section 4 for more details). TensorFlow Privacy [2024] provides the `compute_dp_sgd_privacy_statement` method for computing the privacy parameters and reminds users about the implicit assumption of Poisson subsampling. Ponomareva et al. [2023] note in their survey that "It is common, though inaccurate, to train without Poisson subsampling, but to report the stronger DP bounds as if amplification was used."

As DP-SGD is being deployed in more applications with such discrepancy, it has become crucial to understand exactly how the privacy guarantee depends on the precise choice of the batch generator, especially when one cares about specific (ε, δ) privacy parameters (and not asymptotic bounds). This leads us to our main motivating question:

How do the privacy guarantees of $\mathsf{ABLQ}_{\mathcal{G}}$ *compare for different batch generators* $\mathcal{G}$*?*

Our Contributions. Our main contributions can be summarized as follows:

- In Section 3, we provide a systematic comparison of the DP guarantees for the Deterministic, Shuffle, and Poisson batch generators. Specifically, we demonstrate that, for parameter settings of interest, $\mathsf{ABLQ}_{\mathcal{P}}$ enjoys a stronger DP guarantee than that of $\mathsf{ABLQ}_{\mathcal{S}}$ (see Figure 1); in other words, the practice of implementing the latter while performing privacy analysis for the former can result in incorrect DP claims.
- To address the above issue, in Section 4, we provide a simple adaptation of the Poisson batch generator, with an additional "truncation" step. We show that this *Truncated Poisson Batch Generator* results in nearly the same privacy guarantee (and utility) as the Poisson Batch Generator, but can be easily implemented at scale.
- We introduce a novel *Balls-and-Bins* generator $\mathcal{B}$ (Algorithm 7), which operates by placing each example in a random batch (Section 5). This sampler exhibits behavior similar to Shuffle, with each example appearing in precisely one batch, and can thus be efficiently implemented. For the privacy accounting, we use a Monte Carlo method for estimating the privacy parameters. However, naive Monte Carlo methods are inefficient when estimating small values of δ, or when the number of steps T is large. To this end, we develop

importance sampling techniques for the small δ regime and introduce *order statistics sampling* for cases where T is large; we believe the latter is of independent interest.
- We evaluate DP-SGD on some practical datasets and observe that the model utility of $\mathsf{DP\text{-}SGD}_{\mathcal{B}}$ is comparable to that of $\mathsf{DP\text{-}SGD}_{\mathcal{S}}$ at the same noise scale σ. On the other hand, for each setting of parameters used, we use our Monte Carlo estimation method to show that the privacy guarantees of $\mathsf{ABLQ}_{\mathcal{B}}$ are similar/better relative to $\mathsf{ABLQ}_{\mathcal{P}}$, whereas, the privacy guarantees of $\mathsf{ABLQ}_{\mathcal{S}}$ are much worse.

Finally, we note that yet another sampling approach considered in literature is "sampling without replacement", wherein each batch is sampled independently to be a random fixed-size subset of the entire dataset. Lebeda et al. [2024] showed that this sampling approach also results in a significantly worse privacy guarantee as compared to Poisson subsampling.

1. Preliminaries: Differential Privacy

A *mechanism* $\mathcal{M} : \mathcal{X}^* \to \Delta_{\mathcal{O}}$ maps input datasets to distributions over an output space. On input *dataset* $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\mathcal{M}(\boldsymbol{x}) \in \Delta_{\mathcal{O}}$ is a probability distribution over the output space $\mathcal{O}$; we often use $\mathcal{M}(\boldsymbol{x})$ to denote the underlying random variable as well, and we refer to each $x_i \in \mathcal{X}$ as a *record*. Two datasets $\boldsymbol{x}$ and $\boldsymbol{x}'$ are said to be *adjacent*, denoted $\boldsymbol{x} \sim \boldsymbol{x}'$, if, loosely speaking, they "differ in one record"; in particular, we use the "zeroing-out" adjacency as defined shortly. We consider the following notion of (ε, δ)-*differential privacy (DP)*.

Definition 1.1 (Differential Privacy (DP) [Dwork et al., 2006b,a])**.** For $\varepsilon, \delta \geq 0$, a mechanism $\mathcal{M}$ satisfies (ε, δ)-DP if for all adjacent datasets $\boldsymbol{x} \sim \boldsymbol{x}'$, and for any (measurable) event E it holds that $\Pr[\mathcal{M}(\boldsymbol{x}) \in E] \leq e^{\varepsilon} \Pr[\mathcal{M}(\boldsymbol{x}') \in E] + \delta$.

The *privacy loss curve* for any mechanism $\mathcal{M}$ is a function $\delta_{\mathcal{M}} : \mathbb{R}_{\geq 0} \to [0, 1]$, where $\delta_{\mathcal{M}}(\varepsilon)$ is the smallest δ for which $\mathcal{M}$ satisfies (ε, δ)-DP; $\varepsilon_{\mathcal{M}} : [0, 1] \to \mathbb{R}_{\geq 0}$ can be defined similarly. For notational convenience, for batch generator $\mathcal{G}$, we write $\delta_{\mathcal{G}}$ (resp., $\varepsilon_{\mathcal{G}}$) as a shorthand for $\delta_{\mathsf{ABLQ}_{\mathcal{G}}}$ (resp., $\varepsilon_{\mathsf{ABLQ}_{\mathcal{G}}}$).

1.1. Adjacency Notions.

As alluded to earlier, the notion of adjacency is crucial to Definition 1.1. Commonly used adjacency notions are:

Add-Remove adjacency: Datasets $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}^*$ are said to be *add-remove* adjacent if there exists an i such that $\boldsymbol{x}' = \boldsymbol{x}_{-i}$ or vice-versa (where $\boldsymbol{x}_{-i}$ represents the dataset obtained by removing the ith record in $\boldsymbol{x}$).

Substitution adjacency: Datasets $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}^*$ are said to be *substitution* adjacent if there exists an i such that $\boldsymbol{x}'_{-i} = \boldsymbol{x}_{-i}$.

The privacy analysis of DP-SGD is typically done for the Poisson batch generator $\mathcal{P}$ [Abadi et al., 2016, Mironov, 2017], with respect to the *add-remove* adjacency. However, it is impossible to analyze the privacy of $\mathsf{ABLQ}_{\mathcal{D}}$ or $\mathsf{ABLQ}_{\mathcal{S}}$ with respect to the *add-remove* adjacency because the batch generators $\mathcal{D}$ and $\mathcal{S}$ are not well defined when $n \neq b \cdot T$. On the other hand, using the *substitution* adjacency for $\mathcal{D}$ and $\mathcal{S}$ leads to an unfair comparison to $\mathsf{ABLQ}_{\mathcal{P}}$ whose analysis is typically done with respect to the *add-remove* adjacency. Thus, to make a fair comparison, we consider the following adjacency notion (proposed by Kairouz et al. [2021]).

Zero-out adjacency: We augment the input space to be $\mathcal{X}_\perp := \mathcal{X} \cup \{\perp\}$ and extend any adaptive query method $\mathcal{A}$ as $\mathcal{A}(g_1, \ldots, g_t; \perp) = \mathbf{0}$ for all $g_1, \ldots, g_t \in \mathbb{R}^d$. Datasets $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}_\perp^n$ are said to be *zero-out* adjacent if there exists i such that $\boldsymbol{x}_{-i} = \boldsymbol{x}'_{-i}$, and exactly one of $\{x_i, x'_i\}$ is in $\mathcal{X}$ and the other is $\perp$. Whenever we need to specifically emphasize that $x_i \in \mathcal{X}$ and $x'_i = \perp$, we will denote it as $\boldsymbol{x} \to_z \boldsymbol{x}'$. In this notation, $\boldsymbol{x} \sim \boldsymbol{x}'$ if either $\boldsymbol{x} \to_z \boldsymbol{x}'$ or $\boldsymbol{x}' \to_z \boldsymbol{x}$.

The privacy analysis of $\mathsf{ABLQ}_\mathcal{P}$ with respect to zero-out adjacency is the same as that with respect to the add-remove adjacency; it is essentially replacing a record by a "ghost" record that makes the query method always return $\mathbf{0}$. In the rest of this paper, we only consider this zero-out adjacency.

We note that our separations where $\varepsilon_\mathcal{P}(\delta) \ll \varepsilon_\mathcal{S}(\delta)$, such as in Figure 1, also hold under the substitution adjacency. A naïve way to see this is via "group privacy", namely if a mechanism satisfies (ε, δ)-DP with respect to zero-out adjacency, then it satisfies $(2\varepsilon, \delta \cdot (1 + e^\varepsilon))$-$\mathsf{DP}$ (see, e.g., [Vadhan, 2017]); however, tighter accounting methods for $\varepsilon_\mathcal{P}(\delta)$ under substitution adjacency are also known [Koskela et al., 2020].

1.2. **Hockey Stick Divergence & Dominating Pairs.** For probability densities P and Q, we use $\alpha P + \beta Q$ to denote the weighted sum of the corresponding densities. $P \otimes Q$ denotes the product distribution sampled as (u, v) for $u \sim P$, $v \sim Q$, and, $P^{\otimes T}$ denotes the T-fold product distribution $P \otimes \cdots \otimes P$.

For all $\varepsilon \in \mathbb{R}$, the e^ε-*hockey stick divergence* between P and Q is $D_{e^\varepsilon}(P \parallel Q) := \sup_\Gamma P(\Gamma) - e^\varepsilon Q(\Gamma)$. It is immediate to see that a mechanism $\mathcal{M}$ satisfies (ε, δ)-DP iff for all adjacent $\boldsymbol{x} \sim \boldsymbol{x}'$, it holds that $D_{e^\varepsilon}(\mathcal{M}(\boldsymbol{x}) \parallel \mathcal{M}(\boldsymbol{x}')) \le \delta$.

A key concept for obtaining tight DP guarantees is that of dominating pairs [Zhu et al., 2022], which can be defined as follows.

Definition 1.2 (Dominating Pair [Zhu et al., 2022])**.** The density pair (P, Q) *dominates* the density pair (A, B) (denoted $(P, Q) \succcurlyeq (A, B)$) if $D_{e^\varepsilon}(P \parallel Q) \ge D_{e^\varepsilon}(A \parallel B)$ holds for all $\varepsilon \in \mathbb{R}$.[2] For any mechanism $\mathcal{M}$,

- (P, Q) *dominates* a mechanism $\mathcal{M}$ (denoted $(P, Q) \succcurlyeq \mathcal{M}$) if $(P, Q) \succcurlyeq (\mathcal{M}(\boldsymbol{x}), \mathcal{M}(\boldsymbol{x}'))$ for all adjacent $\boldsymbol{x} \to_z \boldsymbol{x}'$.
- Conversely, (P, Q) is *dominated by* $\mathcal{M}$ (denoted $\mathcal{M} \succcurlyeq (P, Q)$) if there exists $\boldsymbol{x} \to_z \boldsymbol{x}'$ such that $(\mathcal{M}(\boldsymbol{x}), \mathcal{M}(\boldsymbol{x}')) \succcurlyeq (P, Q)$.
- (P, Q) *tightly dominates* $\mathcal{M}$ (denoted $(P, Q) \equiv \mathcal{M}$) if $(P, Q) \succcurlyeq \mathcal{M}$ and $\mathcal{M} \succcurlyeq (P, Q)$.

If $(P, Q) \succcurlyeq \mathcal{M}$, then for all $\varepsilon \ge 0$, it holds that $\delta_\mathcal{M}(\varepsilon) \le \max\{D_{e^\varepsilon}(P \parallel Q), D_{e^\varepsilon}(Q \parallel P)\}$. Conversely, if $\mathcal{M} \succcurlyeq (P, Q)$, it holds that $\delta_\mathcal{M}(\varepsilon) \ge \max\{D_{e^\varepsilon}(P \parallel Q), D_{e^\varepsilon}(Q \parallel P)\}$ for all $\varepsilon \ge 0$.[3] Consequently, if $(P, Q) \equiv \mathcal{M}$, then $\delta_\mathcal{M}(\varepsilon) = \max\{D_{e^\varepsilon}(P \parallel Q), D_{e^\varepsilon}(Q \parallel P)\}$. Thus, tightly dominating pairs completely characterize the privacy loss of a mechanism (although they are not guaranteed to exist for all mechanisms).

We highlight that our terminology is slightly different from Zhu et al. [2022] in that they refer to (P, Q) as a *tightly dominating pair* if $\delta_\mathcal{M}(\varepsilon) = D_{e^\varepsilon}(P \parallel Q)$ for all $\varepsilon \in \mathbb{R}$. Such a notion of a tightly dominating pair always exists, but need not correspond to a worst case pair of adjacent datasets, as is the case for even one step of $\mathsf{ABLQ}_\mathcal{P}$. Our notation is

[2] Note that this includes $\varepsilon < 0$.

[3] This follows from the fact that if $(P, Q) \succcurlyeq (A, B)$ then $(Q, P) \succcurlyeq (B, A)$ (see, e.g., [Zhu et al., 2022, Lemma 46]).

asymmetric in defining a dominating pair for a mechanism, allowing a tight characterization of $\mathsf{ABLQ}_{\mathcal{P}}$.

Dominating pairs behave nicely under mechanism compositions: if $(P_1, Q_1) \succcurlyeq \mathcal{M}_1$ and $(P_2, Q_2) \succcurlyeq \mathcal{M}_2$, then $(P_1 \otimes P_2, Q_1 \otimes Q_2) \succcurlyeq \mathcal{M}_1 \circ \mathcal{M}_2$, the (adaptively) composed mechanism.

Finally, we will need the post-processing property of dominating pairs. To state this, for a distribution P over Ω and a randomized function $f : \Omega \to \Gamma$, let $f(P)$ denote the distribution of $f(x)$ for $x \sim P$. The post-processing property of DP can be written as follows.

Lemma 1.3. *For distributions P, Q over Ω, and distributions A, B over Γ, if there exists $f : \Omega \to \Gamma$ such that simultaneously $f(P) = A$ and $f(Q) = B$ then $(P, Q) \succcurlyeq (A, B)$.*[4]

2. Dominating Pairs for $\mathsf{ABLQ}_{\mathcal{G}}$

As discussed in the previous section, tightly dominating pairs completely describe the privacy loss curve of a mechanism. Given this, a natural strategy for tight privacy accounting for $\mathsf{ABLQ}_{\mathcal{G}}$ is to find (tightly) dominating pairs for these mechanisms. Note that $\mathsf{ABLQ}_{\mathcal{G}}$ is not a single mechanism, but a family of them (parameterized by the underlying adaptive query method $\mathcal{A}$); we can extend the notion of "dominating pair" to say that $(P, Q) \succcurlyeq \mathsf{ABLQ}_{\mathcal{G}}$ if (P, Q) dominates every mechanism in the family, and $\mathsf{ABLQ}_{\mathcal{G}} \succcurlyeq (P, Q)$ if some mechanism in the family dominates (P, Q). The focus of this section is to do so for $\mathcal{G} \in \{\mathcal{D}, \mathcal{P}, \mathcal{S}\}$, which will be crucial for establishing our results in the sequel. Our results henceforth will focus on the one epoch setting, i.e., $n = b \cdot T$, and we will omit explicitly stating this assumption for brevity.

2.1. Tightly Dominating Pair for $\mathsf{ABLQ}_{\mathcal{D}}$.

It follows from the standard analysis of the Gaussian mechanism and parallel composition that a tightly dominating pair for $\mathsf{ABLQ}_{\mathcal{D}}$ is the pair $(P_{\mathcal{D}} := \mathcal{N}(1, \sigma^2), Q_{\mathcal{D}} := \mathcal{N}(0, \sigma^2))$, leading to a closed-form expression for $\delta_{\mathcal{D}}(\varepsilon)$:

Proposition 2.1 (Theorem 8 in Balle and Wang [2018])**.** *For all $\sigma > 0$ and $T \geq 1$, it holds that $(P_{\mathcal{D}}, Q_{\mathcal{D}}) \equiv \mathsf{ABLQ}_{\mathcal{D}}$ for $P_{\mathcal{D}} := \mathcal{N}(1, \sigma^2)$ and $Q_{\mathcal{D}} := \mathcal{N}(0, \sigma^2)$.*

Moreover, for all $\varepsilon \geq 0$, it holds that $\delta_{\mathcal{D}}(\varepsilon) = \Phi\left(-\sigma\varepsilon + \frac{1}{2\sigma}\right) - e^{\varepsilon}\Phi\left(-\sigma\varepsilon - \frac{1}{2\sigma}\right)$, where $\Phi(\cdot)$ is the cumulative density function (CDF) of the standard normal distribution $\mathcal{N}(0, 1)$.

2.2. Tightly Dominating Pair for $\mathsf{ABLQ}_{\mathcal{P}}$.

Zhu et al. [2022] showed[5] that the tightly dominating pair for a single step of $\mathsf{ABLQ}_{\mathcal{P}}$, a Poisson sub-sampled Gaussian mechanism, is given by the pair $(A = (1-q)\mathcal{N}(0, \sigma^2) + q\mathcal{N}(1, \sigma^2), B = \mathcal{N}(0, \sigma^2))$, where q is the sub-sampling probability of each record, namely $q = b/n$, and in the case where $n = b \cdot T$, we have $q = 1/T$. Since $\mathsf{ABLQ}_{\mathcal{P}}$ is a T-fold composition of this Poisson subsampled Gaussian mechanism, it follows that the tightly dominating pair for $\mathsf{ABLQ}_{\mathcal{P}}$ is $(P_{\mathcal{P}} := A^{\otimes T}, Q_{\mathcal{P}} := B^{\otimes T})$.

Proposition 2.2 (Koskela et al. [2020], Zhu et al. [2022])**.** *For all $\sigma > 0$ and $T \geq 1$, it holds that $(P_{\mathcal{P}}, Q_{\mathcal{P}}) \equiv \mathsf{ABLQ}_{\mathcal{P}}$ for*

$$\begin{aligned} P_{\mathcal{P}} &:= \left(\left(1 - \tfrac{1}{T}\right)\mathcal{N}(0, \sigma^2) + \tfrac{1}{T}\mathcal{N}(1, \sigma^2)\right)^{\otimes T}, \\ Q_{\mathcal{P}} &:= \mathcal{N}(0, \sigma^2)^{\otimes T}. \end{aligned}$$

[4] More strongly, the converse is also true (Lemma F.4).

[5] This was implicit in prior work, e.g., [Koskela et al., 2020].

The hockey stick divergences $D_{e^\varepsilon}(P_\mathcal{P} \parallel Q_\mathcal{P})$ and $D_{e^\varepsilon}(Q_\mathcal{P} \parallel P_\mathcal{P})$ do not have a closed-form expression, but there are privacy accountants based on the methods of *Rényi-DP* (RDP) [Mironov, 2017] as well as *privacy loss distributions (PLD)* [Meiser and Mohammadi, 2018, Sommer et al., 2019], the latter providing numerically accurate algorithms [Koskela et al., 2020, Gopi et al., 2021, Ghazi et al., 2022, Doroshenko et al., 2022], that have been the basis of multiple open-source implementations from both industry and academia including [Prediger and Koskela, 2020, Google's DP Library., 2020, Microsoft., 2021]. While RDP-based accounting provides an upper bound on $\delta_\mathcal{P}(\varepsilon)$, the PLD-based accounting implementations can provide upper and lower bounds on $\delta_\mathcal{P}(\varepsilon)$ to high accuracy, as controlled by a certain discretization parameter.

2.3. **Tightly Dominating Pair for $\mathsf{ABLQ}_\mathcal{S}$?** It is not clear which adjacent pair would correspond to a tightly dominating pair for $\mathsf{ABLQ}_\mathcal{S}$, and moreover, it is even a priori unclear if one even exists. However, in order to prove *lower bounds* on the privacy parameters, it suffices to consider a specific instantiation of the adaptive query method $\mathcal{A}$, and a particular adjacent pair $\boldsymbol{x} \sim \boldsymbol{x}'$. In particular, we instantiate the query method $\mathcal{A}$ as follows.

Consider the input space $\mathcal{X} = [-1, 1]$, and assume that the query method $\mathcal{A}$ is non-adaptive, and always produces the query $\psi_t(x) = x$. We consider the adjacent datasets:

- $\boldsymbol{x} = (x_1 = -1, \ldots, x_{n-1} = -1, x_n = 1)$, and
- $\boldsymbol{x}' = (x_1 = -1, \ldots, x_{n-1} = -1, x_n = \perp)$.

In this case, when the differing record falls in batch t, then output of the mechanism on all batches $t' \neq t$ is centered at $-b$, and is centered at $-b+2$ (on input $\boldsymbol{x}$) or at $-b+1$ (on input $\boldsymbol{x}'$) on batch t. Thus it follows that the distributions $A = \mathsf{ABLQ}_\mathcal{S}(\boldsymbol{x})$ and $B = \mathsf{ABLQ}_\mathcal{S}(\boldsymbol{x}')$ are given as:

$$\begin{aligned} A &= \textstyle\sum_{t=1}^T \frac{1}{T} \cdot \mathcal{N}(-b \cdot \mathbf{1} + 2e_t, \sigma^2 I_T), \\ B &= \textstyle\sum_{t=1}^T \frac{1}{T} \cdot \mathcal{N}(-b \cdot \mathbf{1} + e_t, \sigma^2 I_T), \end{aligned}$$

where $\mathbf{1}$ denotes the all-ones vector in $\mathbb{R}^T$ and e_t denotes the tth standard basis vector in $\mathbb{R}^T$. Shifting the distributions by $b \cdot \mathbf{1}$ does not change the hockey stick divergence $D_{e^\varepsilon}(A \parallel B)$, hence we might as well consider the pair

$$P_\mathcal{S} := \tfrac{1}{T} \textstyle\sum_{t=1}^T \mathcal{N}(2e_t, \sigma^2 I_T), \tag{2.1}$$

$$Q_\mathcal{S} := \tfrac{1}{T} \textstyle\sum_{t=1}^T \mathcal{N}(e_t, \sigma^2 I_T). \tag{2.2}$$

To summarize, we have $\mathsf{ABLQ}_\mathcal{S} \succcurlyeq (P_\mathcal{S}, Q_\mathcal{S})$, i.e., $\delta_\mathcal{S}(\varepsilon) \geq \max\{D_{e^\varepsilon}(P_\mathcal{S} \parallel Q_\mathcal{S}), D_{e^\varepsilon}(Q_\mathcal{S} \parallel P_\mathcal{S})\}$. We conjecture that this pair in fact dominates for $\mathsf{ABLQ}_\mathcal{S}$ for all instantiations of query methods $\mathcal{A}$ (including adaptive ones). We elaborate more in Section 3.3.1.

Conjecture 2.3. $(P_\mathcal{S}, Q_\mathcal{S}) \succcurlyeq \mathsf{ABLQ}_\mathcal{S}$.

None of the results in this paper rely on this conjecture being true, as we only use the (dominated) pair $(P_\mathcal{S}, Q_\mathcal{S})$ to establish *lower bounds* on $\delta_\mathcal{S}(\cdot)$.

3. Privacy Loss Comparisons of Existing Batch Generators

In this section, we provide comparisons of the privacy loss curves of $\mathsf{ABLQ}_\mathcal{G}$ for $\mathcal{G} \in \{\mathcal{D}, \mathcal{S}, \mathcal{P}\}$.

3.1. **$\mathsf{ABLQ}_{\mathcal{D}}$ vs. $\mathsf{ABLQ}_{\mathcal{S}}$.** We first note that $\mathsf{ABLQ}_{\mathcal{S}}$ enjoys stronger privacy guarantees than $\mathsf{ABLQ}_{\mathcal{D}}$, as formalized below. This follows from a standard technique that shuffling cannot degrade the privacy guarantee satisfied by a mechanism. For completeness, we provide a proof in Appendix C.

Theorem 3.1. *For all $\sigma, \varepsilon \geq 0$ and $T \geq 1$: $\delta_{\mathcal{S}}(\varepsilon) \leq \delta_{\mathcal{D}}(\varepsilon)$.*

3.2. **$\mathsf{ABLQ}_{\mathcal{D}}$ vs. $\mathsf{ABLQ}_{\mathcal{P}}$.** We show that $\mathsf{ABLQ}_{\mathcal{D}}$ and $\mathsf{ABLQ}_{\mathcal{P}}$ have incomparable privacy loss. In particular, we show the following.

Theorem 3.2. *For all $\sigma > 0$ and $T > 1$, there exist $\varepsilon_0, \varepsilon_1 \geq 0$ such that,*
(a) *$\forall \varepsilon \in [0, \varepsilon_0)$, it holds that $\delta_{\mathcal{D}}(\varepsilon) > \delta_{\mathcal{P}}(\varepsilon)$, and*
(b) *$\forall \varepsilon > \varepsilon_1$, it holds that $\delta_{\mathcal{D}}(\varepsilon) < \delta_{\mathcal{P}}(\varepsilon)$.*

We defer the detailed proof to Appendix D, and provide a proof sketch here. Part (a) is shown by first establishing that the total variation distance (corresponds to $D_1(\cdot \parallel \cdot)$) between $P_{\mathcal{D}}$ and $Q_{\mathcal{D}}$ is strictly larger than the total variation distance between $P_{\mathcal{P}}$ and $Q_{\mathcal{P}}$ when $T > 1$ and $\sigma > 0$. This implies that, $\delta_{\mathcal{D}}(0) > \delta_{\mathcal{P}}(0)$. By using the continuity of $D_{e^\varepsilon}(\cdot \parallel \cdot)$ in ε, we conclude the same for all $\varepsilon < \varepsilon_0$.

For part (b), we construct an explicit set E such that $P_{\mathcal{P}}(E) - e^\varepsilon Q_{\mathcal{P}}(E) > \delta_{\mathcal{D}}(\varepsilon)$. In particular, we choose a halfspace $E := \left\{w \in \mathbb{R}^T \middle| \sum_i w_i > (\varepsilon + \log 2 + T \log T)\sigma^2 + \frac{T}{2}\right\}$ and show that $P_{\mathcal{P}}(E) - e^\varepsilon Q_{\mathcal{P}}(E)$ is at least $\frac{1}{2}\Phi\left(-\frac{\varepsilon\sigma}{\sqrt{T}} - \frac{(T\log T + \log 2)\sigma}{\sqrt{T}} - \frac{\sqrt{T}}{2\sigma}\right)$. For large ε, the dominant term is $-\varepsilon\sigma/\sqrt{T}$. On other hand, $\delta_{\mathcal{D}}(\varepsilon)$ is at most $\Phi(-\varepsilon\sigma + \frac{1}{2\sigma})$ (from Proposition 2.1), which has the dominant term $-\varepsilon\sigma$. Since $-\varepsilon\sigma/\sqrt{T}$ decays slower than $-\varepsilon\sigma$, we get that for sufficiently large ε_1, it holds that $\delta_{\mathcal{D}}(\varepsilon) < \delta_{\mathcal{P}}(\varepsilon)$ for all $\varepsilon > \varepsilon_1$.

Even though Theorem 3.2 was proved for some values of ε_0 and ε_1, we conjecture that it holds for $\varepsilon_0 = \varepsilon_1$.

Conjecture 3.3. *Theorem 3.2 holds for $\varepsilon_0 = \varepsilon_1$.*

Again, we do *not* rely on this conjecture being true in the rest of this paper. Nevertheless, we provide a numerical example that provides evidence for Conjecture 3.3: In Figure 2, for $\sigma = 0.3$ and $T = 10$, we plot the numerically computed $\delta_{\mathcal{D}}(\varepsilon)$ (using Proposition 2.1), as well as lower and upper bounds on $\delta_{\mathcal{P}}(\varepsilon)$, computed using the open source `dp_accounting` library [Google's DP Library., 2020].

3.3. **$\mathsf{ABLQ}_{\mathcal{P}}$ vs. $\mathsf{ABLQ}_{\mathcal{S}}$.** From Theorems 3.1 and 3.2, it follows that there exists an ε_1 such that for all $\varepsilon > \varepsilon_1$, it holds that $\delta_{\mathcal{S}}(\varepsilon) \leq \delta_{\mathcal{D}}(\varepsilon) < \delta_{\mathcal{P}}(\varepsilon)$. On the other hand, while we know that $\delta_{\mathcal{D}}(\varepsilon) > \delta_{\mathcal{P}}(\varepsilon)$ for sufficiently small ε, this does not imply anything about the relationship between $\delta_{\mathcal{S}}(\varepsilon)$ and $\delta_{\mathcal{P}}(\varepsilon)$ for small ε.

We demonstrate simple numerical settings where $\delta_{\mathcal{S}}(\varepsilon)$ is significantly larger than $\delta_{\mathcal{P}}(\varepsilon)$. We prove *lower bounds* on $\delta_{\mathcal{S}}(\varepsilon)$ by constructing specific sets E and using the fact that $\delta_{\mathcal{S}}(\varepsilon) \geq P_{\mathcal{S}}(E) - e^\varepsilon Q_{\mathcal{S}}(E)$.

In particular, we consider sets E_C parameterized by C of the form $\{w \in \mathbb{R}^T : \max_t w_t \geq C\}$; note that E_C is the complement of T-fold Cartesian product of the set $(-\infty, C)$. For a

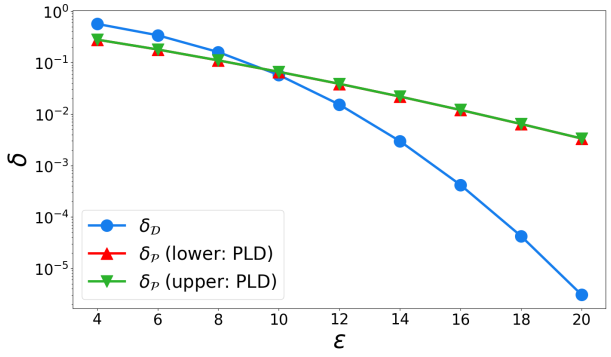

Figure 2: $\delta_{\mathcal{D}}(\varepsilon)$ and $\delta_{\mathcal{P}}(\varepsilon)$ for $\sigma = 0.3$ and $T = 10$.

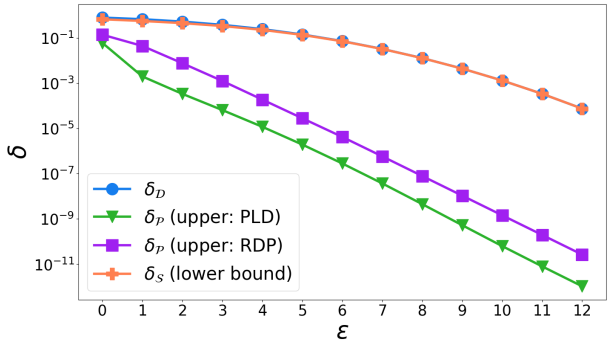

Figure 3: $\delta_{\mathcal{D}}(\varepsilon)$, upper bounds on $\delta_{\mathcal{P}}(\varepsilon)$ and a lower bound on $\delta_{\mathcal{S}}(\varepsilon)$ for varying ε and fixed $\sigma = 0.4$ and $T = 10\,000$.

single Gaussian distribution $D = \mathcal{N}(\mu, \sigma^2 I)$, we can compute the probability mass of E_C under measure D as:

$$\begin{aligned} D(E_C) &= 1 - D(\mathbb{R}^T \setminus E_C) \\ &= 1 - \textstyle\prod_{t=1}^{T} \Pr_{x\sim\mathcal{N}(\mu_t,\sigma^2)}[x < C] \\ &= 1 - \textstyle\prod_{t=1}^{T} \Phi\left(\frac{C-\mu_t}{\sigma}\right). \end{aligned}$$

Specifically, when μ is $\alpha \cdot e_t$ for any standard basis vector e_t, we have $D(E_C) = 1 - \Phi\left(\frac{C-\alpha}{\sigma}\right) \cdot \Phi\left(\frac{C}{\sigma}\right)^{T-1}$. If $D_t = N(2e_t, \sigma^2 I_T)$, we then have that

$$P_{\mathcal{S}}(E_C) = \textstyle\sum_{t=1}^{T} \frac{1}{T} D_t(E_C) = 1 - \Phi\left(\frac{C-2}{\sigma}\right) \cdot \Phi\left(\frac{C}{\sigma}\right)^{T-1}.$$

Similarly, we have $Q_{\mathcal{S}}(E_C) = 1 - \Phi\left(\frac{C-1}{\sigma}\right) \cdot \Phi\left(\frac{C}{\sigma}\right)^{T-1}$.

Thus, we use the following lower bound:

$$\delta_{\mathcal{S}}(\varepsilon) \geq \max_{C\in\mathcal{C}} P_{\mathcal{S}}(E_C) - e^{\varepsilon}, Q_{\mathcal{S}}(E_C) \tag{3.1}$$

for any suitable set $\mathcal{C}$ that can be enumerated over. In our experiments described below, we set $\mathcal{C}$ to be the set of all values of C ranging from 0 to 100 in increments of 0.01.

In Figure 3, we set $\sigma = 0.4$ and number of steps $T = 10\,000$ and plot $\delta_{\mathcal{D}}(\varepsilon)$, an upper bound on $\delta_{\mathcal{P}}(\varepsilon)$ (obtained using `dp_accounting`) and a lower bound on $\delta_{\mathcal{S}}(\varepsilon)$ as obtained via

(3.1). We find that while $\delta_{\mathcal{P}}(4) \leq 1.18 \cdot 10^{-5}$, $\delta_{\mathcal{S}}(4) \geq 0.226$, that is close to $\delta_{\mathcal{D}}(4) \approx 0.244$. Even $\delta_{\mathcal{S}}(12) \geq 7.5 \cdot 10^{-5}$ is larger than $\delta_{\mathcal{P}}(4)$. While the $(4, 1.2 \cdot 10^{-5})$-DP guarantee of $\mathsf{ABLQ}_{\mathcal{P}}$ could have been considered as sufficiently private, $\mathsf{ABLQ}_{\mathcal{S}}$ only satisfies much worse privacy guarantees. We provide additional examples in Appendix A.

3.3.1. *Intuition for Conjecture 2.3.* We attempt to intuit why $\mathsf{ABLQ}_{\mathcal{S}}$ does not provide as much amplification over $\mathsf{ABLQ}_{\mathcal{D}}$, compared to $\mathsf{ABLQ}_{\mathcal{P}}$, and why we conjecture that the pair $(P_{\mathcal{S}}, Q_{\mathcal{S}})$ dominates $\mathsf{ABLQ}_{\mathcal{S}}$ (Conjecture 2.3).

For sake of intuition, let us consider the setting where the query method $\mathcal{A}$ always generates the query $\psi_t(x) = x$, and we have two adjacent datasets:

- $\boldsymbol{x} = (x_1 = -L, \ldots, x_{n-1} = -L, x_n = 1)$, and
- $\boldsymbol{x}' = (x_1 = -L, \ldots, x_{n-1} = -L, x_n = \perp)$.

The case of $L > 1$ is not valid, since in this case $|\psi_t(x)| = L > 1$. However, we can still ask how well the privacy of the nth example is preserved by $\mathsf{ABLQ}_{\mathcal{G}}$ by considering the hockey stick divergence between $\mathsf{ABLQ}_{\mathcal{G}}(\boldsymbol{x})$ and $\mathsf{ABLQ}_{\mathcal{G}}(\boldsymbol{x}')$.

The crucial difference between $\mathsf{ABLQ}_{\mathcal{P}}$ and $\mathsf{ABLQ}_{\mathcal{S}}$ is that the privacy analysis of $\mathsf{ABLQ}_{\mathcal{P}}$ does not depend at all on the non-differing records in the two datasets. In the case of $\mathsf{ABLQ}_{\mathcal{S}}$, we observe that for any fixed σ and T, the hockey stick divergence $D_{e^\varepsilon}(\mathsf{ABLQ}_{\mathcal{S}}(\boldsymbol{x}) \parallel \mathsf{ABLQ}_{\mathcal{S}}(\boldsymbol{x}'))$ approaches $\delta_{\mathcal{D}}(\varepsilon)$ as $L \to \infty$. We sketch this argument intuitively: For any batch S_t that does not contain n, the corresponding $g_t = -bL + e_t$ for $e_t \sim \mathcal{N}(0, \sigma^2)$. Whereas for the batch S_t that contains n, the corresponding $g_t = -(b-1)L + 1 + e_t$ in case of input $\boldsymbol{x}$ or $g_t \sim -(b-1)L + e_t$ in case of input $\boldsymbol{x}'$. As $L \to \infty$, we can identify the batch S_t that contains n with probability approaching 1, thereby not providing any amplification.

In summary, the main differing aspect about $\mathsf{ABLQ}_{\mathcal{S}}$ and $\mathsf{ABLQ}_{\mathcal{P}}$ is that in the former, the non-differing examples can leak information about the location of the differing example in the shuffled order, but it is not the case in the latter. While we sketched an argument that works asymptotically as $L \to \infty$, we see glimpses of it already at $L = 1$.

Our intuitive reasoning for Conjecture 2.3 is that even in the case of (vector-valued) adaptive query methods, in order to "leak the most information" about the differing record x_i between $\boldsymbol{x}$ and $\boldsymbol{x}'$, it seems natural that the query $\psi_t(x_j)$ should evaluate to $-\psi_t(x_i)$ for all $j \neq i$. If the query method satisfies this condition for all t, then it is easy to show that $(P_{\mathcal{S}}, Q_{\mathcal{S}}) \equiv (\mathsf{ABLQ}_{\mathcal{S}}(\boldsymbol{x}), \mathsf{ABLQ}_{\mathcal{S}}(\boldsymbol{x}'))$. Conjecture 2.3 asserts this is indeed the worst case.

4. Truncated Poisson Batch Generator

A key challenge in implementing DP-SGD with Poisson subsampling at scale is that each batch size $|S_t|$ can be different (and random). Variable batches are typically inconvenient to handle in deep learning systems. For example, upon a change in the input shape, `jax.jit` triggers a recompilation of the computation graph, and `tf.function` will retrace the computation graph. Additionally, Google TPUs require all operations to have fixed input and output shapes. To this end, we introduce *truncated Poisson subsampling* to circumvent variable batch sizes. In particular, we choose an upper bound on the maximum batch size B that our training can handle, and given any variable size batch b, if $b \geq B$, we randomly sub-select B examples to retain in the batch, and if $b < B$, we pad the batch with $B - b$ dummy examples with *zero* weight. This deviates slightly from standard Poisson subsampling since our batch sizes can never exceed B. We choose B to be sufficiently larger than the expected batch size,

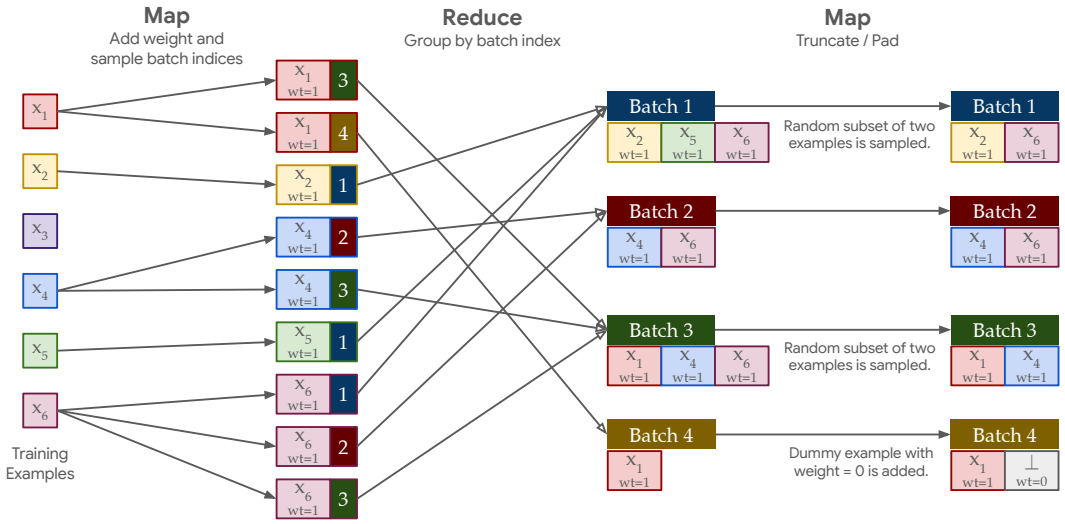

Figure 4: Visualization of the Map-Reduce approach for Poisson subsampling at scale. Consider records $x_1, \ldots, x_6$ sub-sampled into 4 batches with a maximum batch size of $B = 2$. The `Map` operation adds a "weight" parameter of 1 to all examples, and samples indices of batches to which each example will belong. The `Reduce` operation groups by the batch indices. The final `Map` operation truncates batches with more than B examples (e.g., batches $1, 3$ above), and pads dummy examples with weight 0 in batches with fewer than B examples (e.g., batch 4 above).

so that the probability that the sampled batch size b exceeds the maximum allowed batch size B is small. Generating these truncated Poisson subsampled batches can be difficult when the dataset is too large to fit in memory. We use the Map-Reduce framework [Dean and Ghemawat, 2008] to generate batches with truncated Poisson subsampling in a scalable manner as visualized in Figure 4; the details, with a `beam` [The Apache Software Foundation, 2025a] pipeline implementation, are provided in Appendix E.

4.1. **Privacy Analysis.** Our proposed modification can be modeled in terms of ABLQ where the batch generator is a modified version of the Poisson batch generator (Algorithm 5) such that, if a batch size $|S_t|$ is larger than B, then we reduce its size by picking an arbitrary B-size subset of S_t (specifically, we sample a random subset of S_t of size B, but we do not use this aspect in the privacy analysis). This is described in full in Algorithm 6, which we refer to as the *Truncated Poisson Batch Generator* and denote it by $\mathcal{P}_{b,B,T}$. Note that if $B \geq n$, then $\mathcal{P}_{b,B,T}$ is exactly the same as the Poisson batch generator. However, for $B < n$, the two generators are not the same. Below we provide a modification to the analysis of $\mathsf{ABLQ}_{\mathcal{P}}$.

In order to handle the difference of the two mechanisms, we use the following standard proposition. Note that $D_1(P \parallel P')$ corresponds to the *statistical distance* between P and P'.

Proposition 4.1. *For distributions P, P', Q, Q' such that $D_1(P \parallel P'), D_1(Q \parallel Q') \leq \eta$, and $D_{e^\varepsilon}(P' \parallel Q') \leq \delta$, then $D_{e^\varepsilon}(P \parallel Q) \leq \delta + \eta(1 + e^\varepsilon)$.*

Algorithm 6: $\mathcal{P}_{b,B,T}$: Truncated Poisson Batch Generator.

Parameters: Expected batch size b, maximum batch size B, number of batches T.
Input: Number of datapoints n.
Output: Sequence of batches $S_1, \ldots, S_T \subseteq [n]$, with $|S_t| \leq B$.
for $t = 1, \ldots, T$ **do**
  $S_t \leftarrow \emptyset$
  for $i = 1, \ldots, n$ **do**
    $S_t \leftarrow \begin{cases} S_t \cup \{i\} & \text{with probability } b/n \\ S_t & \text{with probability } 1 - b/n \end{cases}$
  end
  if $|S_t| > B$ **then**
    $S_t \leftarrow$ arbitrary (e.g. random) subset of S_t of size B
  end
end
return $S_1, \ldots, S_T$

Proof. For any event Γ we have that

$$P(\Gamma) \overset{\text{(i)}}{\leq} P'(\Gamma) + \eta \overset{\text{(ii)}}{\leq} e^{\varepsilon} Q'(\Gamma) + \delta + \eta \overset{\text{(iii)}}{\leq} e^{\varepsilon}(Q(\Gamma) + \eta) + \delta + \eta = e^{\varepsilon} Q(\Gamma) + \delta + \eta(1 + e^{\varepsilon}),$$

where (i) follows from $D_1(P \parallel P') \leq \eta$, (ii) follows from $D_{e^{\varepsilon}}(P' \parallel Q') \leq \delta$ and (iii) follows from $D_1(Q \parallel Q') \leq \eta$. Thus, we get that $D_{e^{\varepsilon}}(P \parallel Q) \leq \delta + \eta(1 + e^{\varepsilon})$. □

The batch size $|S_t|$ before truncation in $\mathcal{P}_{b,B,T}$ is distributed as the binomial distribution $\mathrm{Bin}(n, b/n)$, and thus, by a union bound over the events that the sampled batch size $|S_t| > B$ at any step, it follows that for any input dataset $\boldsymbol{x}$,

$$D_1(\mathsf{ABLQ}_{\mathcal{P}_{b,B,T}}(\boldsymbol{x}) \parallel \mathsf{ABLQ}_{\mathcal{P}}(\boldsymbol{x})) \leq T \cdot \Psi(n, b, B),$$

where $\Psi(n, b, B) := \Pr_{r \sim \mathrm{Bin}(n, b/n)}[r > B]$. Applying Proposition 4.1 we get

Theorem 4.2. *For all $\sigma, b > 0$, $\varepsilon \geq 0$, and positive integers $n \geq b$, B, T, it holds that*

$$\delta_{\mathcal{P}_{b,B,T}}(\varepsilon) \leq \max\{D_{e^{\varepsilon}}(P_{\mathcal{P}} \parallel Q_{\mathcal{P}}), D_{e^{\varepsilon}}(Q_{\mathcal{P}} \parallel P_{\mathcal{P}})\} + T \cdot (1 + e^{\varepsilon}) \cdot \Psi(n, b, B).$$

Note that $\Psi(n, b, B)$ can be made arbitrarily small by increasing B, which affects the computation cost. In particular, given a target privacy parameter (ε, δ), we can, for example, work backwards to first choose B such that $\Psi(n, b, B) \cdot T \cdot (1 + e^{\varepsilon}) \leq 10^{-5} \cdot \delta$, and then choose the noise scale σ such that $\max\{D_{e^{\varepsilon}}(P_{\mathcal{P}} \parallel Q_{\mathcal{P}}), D_{e^{\varepsilon}}(Q_{\mathcal{P}} \parallel P_{\mathcal{P}})\} \leq (1 - 10^{-5}) \cdot \delta$, using aforementioned privacy accounting libraries.

5. Balls-and-Bins Generator

We next introduce the *Balls-and-Bins* generator $\mathcal{B}_T$ (Algorithm 7), which operates by placing each example in a random batch. An advantage of Balls-and-Bins is that it can be implemented trivially given any implementation of Shuffling. Namely, given examples $x_1, \ldots, x_n$ in a randomly shuffled order, we construct batches of sizes $b_1, \ldots, b_{T-1}, b_T$ in sequential order where each b_t is inductively sampled from the binomial distribution $\mathrm{Bin}(n - \sum_{i=1}^{t-1} b_i, 1/(T - t + 1))$. An alternative approach could be to combinatorially simulate throwing n balls into T bins to generate the sequence of batch sizes, which while less efficient,

Algorithm 7: $\mathcal{B}_T$: Balls-and-Bins Sampler.

Input: Number of datapoints n.
Output: Sequence of batches $S_1, \ldots, S_T \subseteq [n]$.
for $t = 1, \ldots, T$ **do**
  $S_t \leftarrow \emptyset$
end
for $i = 1, \ldots, n$ **do**
  Sample t uniformly at random from $[T]$
  $S_t \leftarrow S_t \cup \{i\}$
end
return $S_1, \ldots, S_T$

potentially avoids floating point errors in the binomial probabilities. Thus, Balls-and-Bins is similar to implement as Shuffling and allows for efficient implementation.

It turns out that the privacy accounting for $\mathsf{ABLQ}_{\mathcal{B}}$ is quite challenging and we devote the remainder of this section to this task.

5.1. **Tightly Dominating Pair for $\mathsf{ABLQ}_{\mathcal{B}}$.** We start by identifying a tightly dominating pair for $\mathsf{ABLQ}_{\mathcal{B}}$:

Theorem 5.1. *For all $\sigma > 0$ and $T \geq 1$, it holds that $(P_{\mathcal{B}}, Q_{\mathcal{B}}) \equiv \mathsf{ABLQ}_{\mathcal{B}}$ for*[6]

$$P_{\mathcal{B}} := \tfrac{1}{T}\textstyle\sum_{t=1}^{T} \mathcal{N}(e_t, \sigma^2 I_T)\,, \quad Q_{\mathcal{B}} := \mathcal{N}(0, \sigma^2 I_T)\,.$$

The proof relies on the following well-known "joint convexity" property of the hockey stick divergence, whose proof we include in Appendix B for completeness.

Lemma 5.2 (Joint Convexity of Hockey Stick Divergence)**.** *Given mixture distributions $P = \sum_{i=1}^{m} \alpha_i P_i$ and $Q = \sum_{i=1}^{m} \alpha_i Q_i$, it holds for all $\varepsilon \in \mathbb{R}$ that $D_{e^\varepsilon}(P \parallel Q) \leq \sum_i \alpha_i D_{e^\varepsilon}(P_i \parallel Q_i)$.*

Proof of Theorem 5.1. To show $\mathsf{ABLQ}_{\mathcal{B}} \succcurlyeq (P_{\mathcal{B}}, Q_{\mathcal{B}})$, we need to only consider a specific adaptive query method $\mathcal{A}$. We consider $\mathcal{X} = [-1, 1]$ and let $\boldsymbol{x} = (0, \ldots, 0, 1)$ and $\boldsymbol{x}' = (0, \ldots, 0, \perp)$. Consider $\mathcal{A}$ that always generates the query $\psi_t(x) = x$ (and by definition $\psi_t(\perp) = 0$). In this case, it is immediate that $P_{\mathcal{B}} = \mathsf{ABLQ}_{\mathcal{B}}(\boldsymbol{x})$ and $Q_{\mathcal{B}} = \mathsf{ABLQ}_{\mathcal{B}}(\boldsymbol{x}')$.

To show that $(P_{\mathcal{B}}, Q_{\mathcal{B}}) \succcurlyeq \mathsf{ABLQ}_{\mathcal{B}}$, consider any underlying adaptive query method $\mathcal{A} : (\mathbb{R}^d)^* \times \mathcal{X} \to \mathbb{B}^d$, and consider any adjacent datasets $\boldsymbol{x} \to_z \boldsymbol{x}'$ that differ on say example x_n and $x'_n = \perp$. Let $\Gamma = (S'_1, \ldots, S'_T)$ where $S'_i \subseteq [n-1]$ be an assignment of batches for all other examples and let $\mathcal{B}_\Gamma$ refer to the batch generator that samples t uniformly in $[T]$ and returns $S_t = S'_t \cup \{n\}$ and $S_r = S'_r$ for all $r \neq t$. We provide a post-processing function f such that $f(P_{\mathcal{B}}) = \mathsf{ABLQ}_{\mathcal{B}_\Gamma}(\boldsymbol{x})$ and $f(Q_{\mathcal{B}}) = \mathsf{ABLQ}_{\mathcal{B}_\Gamma}(\boldsymbol{x}')$ for any Γ. Since $\mathsf{ABLQ}_{\mathcal{B}}(\cdot) = \frac{1}{T^{n-1}} \sum_\Gamma \mathsf{ABLQ}_{\mathcal{B}_\Gamma}(\cdot)$, it follows from Lemmas 1.3 and 5.2 that $(P_{\mathcal{B}}, Q_{\mathcal{B}}) \succcurlyeq \mathsf{ABLQ}_{\mathcal{B}}$.

Consider the randomized post-processing function f that maps vectors $v \in \mathbb{R}^T$ to $(\mathbb{R}^d)^T$ (output space of $\mathsf{ABLQ}_{\mathcal{B}}$) as follows: Given vector $(v_1, \ldots, v_T) \in \mathbb{R}^T$, let $g_t = \sum_{i \in S'_t} \psi_t(x_i) + \psi_t(x_n) \cdot v_t + e_t$ for $e_t \sim \mathcal{N}(0, \sigma^2(I_T - \psi_t(x_n)\psi_t(x_n)^\top))$; this is inductively defined since ψ_t potentially depends on $g_1, \ldots, g_{t-1}$.

[6] Incidentally, the same pair $(P_{\mathcal{B}}, Q_{\mathcal{B}})$ was studied in the context of shuffling by Koskela et al. [2023], who also discussed the difficulty of numerically approximating its hockey stick divergence.

First let us consider the case when $v \sim Q_{\mathcal{B}} = \mathcal{N}(0, \sigma^2 I_T)$. In this case, $\psi_t(x_n)v_t + e_t$ is distributed precisely as $\mathcal{N}(0, \sigma^2 I_T)$, and thus g_t is distributed as $\sum_{i \in S'_t} \psi_t(x_i) + e'_t$ for $e'_t \sim \mathcal{N}(0, \sigma^2 I_T)$ precisely as in $\mathsf{ABLQ}_{\mathcal{B}}(\boldsymbol{x}')$, since $\psi_t(x'_n) = 0$ so it does not matter which batch x'_n lands in. Thus, $f(Q_{\mathcal{B}}) = \mathsf{ABLQ}_{\mathcal{B}}(\boldsymbol{x}')$.

On the other hand, $v \sim P_{\mathcal{B}}$, is equivalent to sampling t_* uniformly at random in $[T]$, and sampling $v \sim \mathcal{N}(e_{t_*}, \sigma^2 I_T)$. For a fixed t_* and sampling $v \sim \mathcal{N}(e_{t_*}, \sigma^2 I_T)$, g_t is distributed as $\sum_{i \in S'_t} \psi_t(x_i) + e'_t$ for $e'_t \sim \mathcal{N}(0, \sigma^2 I_T)$ for $t \neq t_*$ and is distributed as $\sum_{i \in S'_t} \psi_t(x_i) + \psi_t(x_n) + e'_t$ for $e'_t \sim \mathcal{N}(0, \sigma^2 I_T)$ for $t = t_*$, which is precisely as in $\mathsf{ABLQ}_{\mathcal{B}}(\boldsymbol{x})$ when x_n lands in batch S_{t_*}. Since t_* is uniformly random in $[T]$, we get that $f(P_{\mathcal{B}}) = \mathsf{ABLQ}_{\mathcal{B}}(\boldsymbol{x})$. □

5.2. **Privacy Loss Comparisons.** We now compare the privacy of $\mathsf{ABLQ}_{\mathcal{B}}$ against $\mathsf{ABLQ}_{\mathcal{G}}$ for $\mathcal{G} \in \{\mathcal{D}, \mathcal{S}, \mathcal{P}\}$.

5.2.1. $\mathsf{ABLQ}_{\mathcal{B}}$ *vs.* $\mathsf{ABLQ}_{\mathcal{D}}$ *and* $\mathsf{ABLQ}_{\mathcal{S}}$. A simple consequence of Lemma 5.2 is that $\mathsf{ABLQ}_{\mathcal{B}}$ and $\mathsf{ABLQ}_{\mathcal{S}}$ have better privacy guarantees than $\mathsf{ABLQ}_{\mathcal{D}}$, since any fixed assignment of examples to batches results in the analysis being equivalent to $\mathsf{ABLQ}_{\mathcal{D}}$. In fact, more strongly, we observe that in fact $\mathsf{ABLQ}_{\mathcal{B}}$ always has better privacy guarantees than $\mathsf{ABLQ}_{\mathcal{S}}$.

Proposition 5.3. *For all $\varepsilon > 0$, $\sigma > 0$, and $T \geq 1$, it holds that $\delta_{\mathcal{B}}(\varepsilon) \leq \delta_{\mathcal{S}}(\varepsilon) \leq \delta_{\mathcal{D}}(\varepsilon)$.*

Proof. Consider the same case as in the proof of Theorem 5.1 where $\mathcal{X} = [-1, 1]$ and let $\boldsymbol{x} = (0, \ldots, 0, 1)$ and $\boldsymbol{x}' = (0, \ldots, 0, \perp)$. Consider $\mathcal{A}$ that always generates the query $\psi_t(x) = x$ (and by definition $\psi_t(\perp) = 0$). In this case, it is immediate to see that $P_{\mathcal{B}} = \mathsf{ABLQ}_{\mathcal{B}}(\boldsymbol{x}) = \mathsf{ABLQ}_{\mathcal{S}}(\boldsymbol{x})$ and $Q_{\mathcal{B}} = \mathsf{ABLQ}_{\mathcal{B}}(\boldsymbol{x}') = \mathsf{ABLQ}_{\mathcal{S}}(\boldsymbol{x}')$. Thus, we get that $\mathsf{ABLQ}_{\mathcal{S}} \succcurlyeq (P_{\mathcal{B}}, Q_{\mathcal{B}})$ and hence $\delta_{\mathcal{B}}(\varepsilon) \leq \delta_{\mathcal{S}}(\varepsilon)$. □

5.2.2. $\mathsf{ABLQ}_{\mathcal{B}}$ *vs.* $\mathsf{ABLQ}_{\mathcal{P}}$. Finally, we observe that $\mathsf{ABLQ}_{\mathcal{B}}$ has better privacy guarantees than $\mathsf{ABLQ}_{\mathcal{P}}$ at large ε.

Theorem 5.4. *For all $\sigma > 0$ and $T > 1$, there exist $\varepsilon_0 > 0$ such that, $\forall \varepsilon > \varepsilon_0$, it holds that $\delta_{\mathcal{B}}(\varepsilon) < \delta_{\mathcal{P}}(\varepsilon)$.*

Proof. From Theorem 3.2, there exists an ε_0 such that for all $\varepsilon > \varepsilon_0$, it holds that $\delta_{\mathcal{D}}(\varepsilon) < \delta_{\mathcal{P}}(\varepsilon)$. Combining this with Proposition 5.3 gives us that $\delta_{\mathcal{B}}(\varepsilon) < \delta_{\mathcal{P}}(\varepsilon)$ for all $\varepsilon \geq \varepsilon_0$. □

Remark 5.5. *In Appendix F, we show that in fact for all $\sigma > 0$ and $T > 1$, $(P_{\mathcal{B}}, Q_{\mathcal{B}}) \not\succcurlyeq (P_{\mathcal{P}}, Q_{\mathcal{P}})$; the reverse direction $(P_{\mathcal{P}}, Q_{\mathcal{P}}) \not\succcurlyeq (P_{\mathcal{B}}, Q_{\mathcal{B}})$ is already implied by Theorem 5.4. Thus, the privacy guarantees of $\mathsf{ABLQ}_{\mathcal{P}}$ and $\mathsf{ABLQ}_{\mathcal{B}}$ are in general incomparable.*

5.3. **Estimating $\delta_{\mathcal{B}}(\varepsilon)$.** The hockey stick divergence $D_\varepsilon(P \parallel Q)$ can be expressed in terms of the *privacy loss function* $L_{P\|Q}(x)$ [Dwork and Rothblum, 2016][7] as

$$D_{e^\varepsilon}(P \parallel Q) = \mathbb{E}_{x\sim P}\left[1 - e^{\varepsilon - L_{P\|Q}(x)}\right]_+, \tag{5.1}$$

where $L_{P\|Q}(x) := \log \frac{P(x)}{Q(x)}$,[8] where $P(x)$ and $Q(x)$ refers to the density of P and Q at x and $[z]_+ := \max\{0, z\}$. The privacy loss function $L_{P_{\mathcal{B}}\|Q_{\mathcal{B}}}(x)$ for the pair $(P_{\mathcal{B}}, Q_{\mathcal{B}})$ and $(Q_{\mathcal{B}}, P_{\mathcal{B}})$ at $x \in \mathbb{R}^T$ are as follows:

$$\begin{aligned} L_{P_{\mathcal{B}}\|Q_{\mathcal{B}}}(x) &:= \log \frac{P_{\mathcal{B}}(x)}{Q_{\mathcal{B}}(x)} = \log \frac{\sum_{t=1}^T e^{-\|x-e_t\|^2/(2\sigma^2)}}{T\cdot e^{-\|x\|^2/(2\sigma^2)}} \\ &= \log\left(\sum_{t=1}^T e^{x_t/\sigma^2}\right) - \log T - \tfrac{1}{2\sigma^2}, \qquad (5.2) \\ L_{Q_{\mathcal{B}}\|P_{\mathcal{B}}}(x) &= \log T + \tfrac{1}{2\sigma^2} - \log\left(\sum_{t=1}^T e^{x_t/\sigma^2}\right). \qquad (5.3) \end{aligned}$$

5.3.1. *Monte Carlo Estimation.* (5.1) suggests a natural approach for estimating $D_{e^\varepsilon}(P \parallel Q)$, via drawing multiple samples $x \sim P$ and returning the average value of $[1 - e^{\varepsilon - L_{P\|Q}(x)}]_+$; such an approach was previously studied by Wang et al. [2023]. We can obtain high probability upper bounds via a Chernoff–Hoeffding bound as described in Algorithm 8, wherein $\mathsf{KL}(q \parallel p) := q \log \frac{q}{p} + (1-q)\log\frac{1-q}{1-p}$ is the KL divergence for Bernoulli random variables $\mathsf{Ber}(p)$ and $\mathsf{Ber}(q)$.

Theorem 5.6. *For all distributions P and Q, Algorithm 8 returns an upper bound on $D_{e^\varepsilon}(P \parallel Q)$ with probability $1-\beta$.*

Fact 5.7 (Hoeffding [1963])**.** *For $Z_1, \ldots, Z_m$ drawn i.i.d. from distribution P over $[0,1]$ with mean μ,*

$$\Pr\left[\tfrac{1}{m}\textstyle\sum_{i=1}^m Z_i < q\right] \le e^{-\mathsf{KL}(q \parallel \mu)m}.$$

Proof of Theorem 5.6. Suppose $D_{e^\varepsilon}(P \parallel Q) = \mu$. If the returned value p by Algorithm 8 is 1, then $\mu \le 1$ holds trivially. Otherwise if p is smaller than μ, then we have that the empirical average q is such that $\mathsf{KL}(q \parallel \mu) > \mathsf{KL}(q \parallel p) \ge \log(1/\beta)/m$. From Fact 5.7, this can happen with probability at most β. □

Algorithm 8 in principle allows us to obtain an upper bound on the hockey stick divergence to an arbitrary accuracy and with arbitrarily high probability through the guarantees in Theorem 5.6 as the number of samples $m \to \infty$. However, there are two challenges when implementing it in practice. First, the sample size m needed can be quite large if we want a small multiplicative approximation in a regime where $D_{e^\varepsilon}(P \parallel Q)$ is very small. Furthermore in the case of $(P_{\mathcal{B}}, Q_{\mathcal{B}})$, sampling each $x^{(i)}$, which is T dimensional, is computationally intensive for large T. We tackle each of these challenges using importance sampling and a new "order statistics sampling" method respectively; we believe the latter could be of independent interest.

[7]The distribution of $L_{P\|Q}(x)$ for $x \sim P$ is also known as the *privacy loss random variable*. However for later convenience we use the *loss function* terminology.

[8]Technically speaking, $\frac{P(x)}{Q(x)}$ should be replaced by the Radon–Nikodym derivative of $\frac{dP}{dQ}(x)$, but for purposes of this work, it suffices to consider the case of densities.

Algorithm 8: Monte Carlo Estimation of $D_{e^\varepsilon}(P \parallel Q)$.

Input: Distributions P and Q; sample access to P, Sample size m, Error probability β.
Output: An upper confidence bound on $D_{e^\varepsilon}(P \parallel Q)$.
Sample $x^{(1)}, \ldots, x^{(m)} \sim P$
$q \leftarrow \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - e^{\varepsilon - L_{P\parallel Q}(x^{(i)})}\}$
$p \leftarrow$ smallest value in $[q, 1]$ such that $\mathsf{KL}(q \parallel p) \geq \log(1/\beta)/m$, or 1 if no such value exists
return p

Before describing these methods, we first observe a simplification when estimating $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$ using Algorithm 8. For $P_t := \mathcal{N}(e_t, \sigma^2 I)$, we have that $P_{\mathcal{B}} = \frac{1}{T}\sum_{t=1}^{T} P_t$. By symmetry of $L_{P_{\mathcal{B}}\parallel Q_{\mathcal{B}}}(x)$, it follows that

$$\begin{aligned} D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}}) &= \tfrac{1}{T}\textstyle\sum_{t=1}^{T} \mathbb{E}_{x\sim P_t}[1 - e^{\varepsilon - L_{P_{\mathcal{B}}\parallel Q_{\mathcal{B}}}(x)}]_+ \\ &= \mathbb{E}_{x\sim P_1}[1 - e^{\varepsilon - L_{P_{\mathcal{B}}\parallel Q_{\mathcal{B}}}(x)}]_+ \end{aligned}$$

Thus, it suffices to sample $x^{(i)} \sim P_1$ instead of $P_{\mathcal{B}}$ in Algorithm 8.

5.3.2. *Importance Sampling.* We provide a generic approach for improving the sample complexity of Algorithm 8 via importance sampling. For any P and Q, let E_ε be any event such that $L_{P\parallel Q}(x) < \varepsilon$ for all $x \notin E_\varepsilon$. Then, it suffices to "zoom in" on E_ε by sampling from the conditional distribution $P|_{x\in E_\varepsilon}$, when estimating the expectation in (5.1), as explained in Algorithm 9. Since this requires sample access to $P|_{x\in E_\varepsilon}$, the set E_ε has to be chosen carefully so that it will be efficient to sample from $P|_{x\in E_\varepsilon}$. We now define the specific sets E_ε we use for estimating $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$ and $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$, starting with the latter.

For estimating $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$, let $E_\varepsilon := \{x \in \mathbb{R}^T : \max_{t\in[T]} x_t \leq C_\varepsilon\}$ where $C_\varepsilon = \frac{1}{2} + \sigma^2 \cdot (\log T - \varepsilon)$. The choice of C_ε is such that for all $x \notin E_\varepsilon$,

$$\begin{aligned} L_{Q_{\mathcal{B}}\parallel P_{\mathcal{B}}}(x) &= \log T + \tfrac{1}{2\sigma^2} - \log\left(\textstyle\sum_{t=1}^{T} e^{x_t/\sigma^2}\right) \\ &< \log T + \tfrac{1}{2\sigma^2} - \log\left(e^{C_\varepsilon/\sigma^2}\right) \quad \ldots \text{since } \max_t x_t > C_\varepsilon \\ &= \log T + \tfrac{1}{2\sigma^2} - \tfrac{C_\varepsilon}{\sigma^2} = \varepsilon. \end{aligned}$$

We show how to efficiently sample from $Q_{\mathcal{B}}|_{x\in E_\varepsilon}$ in Appendix G.1.

For estimating $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$, we consider $E_\varepsilon := \{x : \max\{x_1 - 1, \max_{t>1} x_t\} \geq C_\varepsilon\}$ such that

$$C_\varepsilon = \tfrac{1}{2} + \sigma^2 \cdot \left(\varepsilon - \log\left(1 + \tfrac{e^{1/\sigma^2}-1}{T}\right)\right).$$

The choice of C_ε implies that for all $x \notin E_\varepsilon$, it holds that $x_1 < C_\varepsilon + 1$ and $x_t < C_\varepsilon$ for all $t > 1$, and hence

$$\begin{aligned} L_{P_{\mathcal{B}}\parallel Q_{\mathcal{B}}}(x) &= \log\left(\textstyle\sum_{t=1}^{T} e^{x_t/\sigma^2}\right) - \log T - \tfrac{1}{2\sigma^2} \\ &< \log(e^{(C_\varepsilon+1)/\sigma^2} + (T-1)e^{C_\varepsilon/\sigma^2}) - \log T - \tfrac{1}{2\sigma^2} \\ &= \tfrac{C_\varepsilon}{\sigma^2} + \log(e^{1/\sigma^2} + T - 1) - \log T - \tfrac{1}{2\sigma^2} \leq \varepsilon. \end{aligned}$$

Algorithm 9: Monte Carlo Estimation of $D_{e^\varepsilon}(P \parallel Q)$ with Importance Sampling.

Input: Distributions P and Q, Event E such that $L_{P\|Q}(x) < \varepsilon$ for all $x \notin E$, Sample access to $P|_{x\in E}$, Sample size m, Error probability β.
Output: An upper confidence bound on $D_{e^\varepsilon}(P \parallel Q)$.
Sample $x^{(1)}, \ldots, x^{(m)} \sim P|_{x\in E}$
$q \leftarrow \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - e^{\varepsilon - L_{P\|Q}(x^{(i)})}\}$
$p \leftarrow$ smallest value in $[q, 1]$ such that $\mathsf{KL}(q \parallel p) \geq \log(1/\beta)/m$, or 1 if no such value exists
return $p \cdot P(E)$

Note that as before we sample from $P_1|_{x\in E_\varepsilon}$ instead of $P_{\mathcal{B}}|_{x\in E_\varepsilon}$. Again, it is efficient to sample from $P_1|_{x\in E_\varepsilon}$, and we defer the details to Appendix G.1.

In general, this approach for importance sampling reduces the sample complexity by a factor of $1/P(E_\varepsilon)$, and we numerically demonstrate the improved sample complexity for specific examples in Appendix G.1.

5.3.3. *Order Statistics Sampling.* As mentioned before, sampling $x \sim P_1$ or $x \sim Q_{\mathcal{B}}$ can be slow when the number of steps T is large, especially since we also need to draw a large sample of size m. To make this efficient, at a slight cost of obtaining pessimistic bounds on $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$ and $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$, we use the following approximation: For any list $k_1, \ldots, k_r \in \{1, \ldots, R\}$ of increasing indices, the following holds, where we use $k_{r+1} := R$ and $k_0 := 0$ for convenience. Given $x_1, \ldots, x_R \in \mathbb{R}$, let $y^{(1)}, \ldots, y^{(R)}$ denote the same values as x_i's but in sorted order $y^{(1)} \geq \cdots \geq y^{(R)}$. Then,

$$\sum_{t=1}^{R} e^{x_t/\sigma^2} \leq \sum_{i=1}^{r} (k_{i+1} - k_i) \cdot e^{y^{(k_i)}/\sigma^2} \tag{5.4}$$

$$\sum_{t=1}^{R} e^{x_t/\sigma^2} \geq \sum_{i=1}^{r} (k_i - k_{i-1}) \cdot e^{y^{(k_i)}/\sigma^2}, \tag{5.5}$$

where (5.4) additionally requires that $k_1 = 1$. These approximations are inspired by and are a generalization of the bounds introduced by Ben Slimane [2001], which correspond to $k_1 = 1$ and $k_2 = 2$ for (5.4) and $k_1 = 1$ and $k_2 = R$ for (5.5).

Our key idea for efficiency is to directly sample from the joint distribution of order statistics $y^{(k_1)}, \ldots, y^{(k_r)}$. In particular, for any distribution P with efficiently computable inverse of the cumulative density function $\mathsf{CDF}_P^{-1}(\cdot)$, Algorithm 10 efficiently samples order statistics of P, wherein $\mathsf{Beta}(a, b)$ is the Beta distribution over $[0, 1]$. Theorem G.5 (Appendix G.2) establishes the correctness of this algorithm.

When estimating $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$ we sample an upper bound on $L_{Q_{\mathcal{B}}\|P_{\mathcal{B}}}(x)$ as

$$\log T + \tfrac{1}{2\sigma^2} - \log\left(\sum_{i=1}^{r} (k_i - k_{i-1}) \cdot e^{y^{(k_i)}/\sigma^2}\right),$$

for $R = T$, and when estimating $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$ we sample an upper bound on $L_{P_{\mathcal{B}}\|Q_{\mathcal{B}}}(x)$ as

$$\log\left(e^{x_1/\sigma^2} + \sum_{i=1}^{r} (k_{i+1} - k_i) \cdot e^{y^{(k_i)}/\sigma^2}\right) - \log T - \tfrac{1}{2\sigma^2},$$

where we sample $x_1 \sim \mathcal{N}(1, \sigma^2)$ and $y^{(k_1)}, \ldots, y^{(k_r)}$ using Algorithm 10 for $R = T - 1$, and plug them into Algorithm 8. While not immediate, this can also be used in conjunction with importance sampling as in Algorithm 9. We defer the details to Appendix G.3.

Algorithm 10: Sampling Order Statistics of P.

Input: Number R of random variables $\sim P$, Orders $k_1, \dots, k_r \in \{1, \dots, R\}$
Output: $(y^{(k_1)}, \dots, y^{(k_r)})$ jointly distributed as $(k_1, \dots, k_r)$ order statistics of R draws from P
Let $k_0 \leftarrow 0$ **for** $i = 1, \dots, r$ **do**
  $z_i \sim \mathsf{Beta}(R - k_i + 1, k_i - k_{i-1})$
  $y^{(k_i)} \leftarrow \mathsf{CDF}_P^{-1}(\prod_{j=1}^{i} z_j)$
end
return $(y^{(k_1)}, \dots, y^{(k_r)})$

5.3.4. *Lower Bounds on $\delta_{\mathcal{B}}(\varepsilon)$.* Finally, in addition to Monte Carlo estimation, we can obtain a lower bound on $\delta_{\mathcal{B}}(\varepsilon)$ that is efficient to compute, similar to Section 3.3. The idea is to use the following lower bound

$$\delta_{\mathcal{B}}(\varepsilon) \geq D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}}) \geq \sup_C P_{\mathcal{B}}(E_C) - e^\varepsilon Q_{\mathcal{B}}(E_C), \tag{5.6}$$

where $E_C = \{x : \max_t x_t \geq C\}$ is the same as defined in Section 3.3. Again, $P_{\mathcal{B}}(E_C)$ and $Q_{\mathcal{B}}(E_C)$ are efficiently computable. We numerically observe that the lower bounds computed by this method are in fact quite close to the Monte Carlo estimates of $\delta_{\mathcal{B}}(\varepsilon)$ found via Algorithm 8; see, e.g., Figure 6.

Finally, we note that Monte Carlo estimation can be easily parallelized by having different machines generate samples and then combining the estimates, which can reduce the wall clock time. This allows scaling the Monte Carlo estimation to a large number of samples. Our implementation of privacy accounting is available at github.com/google-research/google-research/tree/master/dpsgd_batch_sampler_accounting.

6. Experiments

We compare the utility of DP-SGD using $\mathcal{S}$, $\mathcal{P}$, and $\mathcal{B}$ batch generators. We compare all algorithms with the same noise scale σ to isolate the impact of using different batch generators from the privacy accounting.

Datasets. The first dataset we consider is the Criteo Display Ads pCTR dataset [Jean-Baptiste Tien, 2014], which contains around 46M examples. We split the labeled training set from the dataset chronologically into a 80%/10%/10% partition of train/validation/test sets. We consider the task of predicting the probability of ad click from the remaining features.

The second dataset we consider is the Criteo Sponsored Search Conversion Log dataset [Tallis and Yadav, 2018], which contains 16M examples. We randomly split the dataset into a 80%/20% partition of train/test sets. We consider a conversion prediction task, where we predict the binary feature `Sale`. We omit the features denoted *Outcome/Labels* in Tallis and Yadav [2018], and `product_price`, which is highly correlated with the label.

For both datasets, we use the binary cross entropy loss for training and report the AUC on the labeled test split, averaged over three runs with each run using independently generated batches. We plot the results with error bars indicating a single standard deviation. For more details about the model architectures and training, see Appendix H.

Results. We train with DP-SGD with various values of σ, and for reference, we also train with regular SGD without any clipping or noise for different batch sizes. The model utilities in terms of AUC are in Figure 5. The σ values we chose were motivated as follows:

(i) First, we consider "large" values of σ, namely in $\{0.1, 0.2, 0.3, 0.4\}$, such that the privacy parameters when using Poisson subsampling / Ball-and-Bins sampling are in a regime that is common in practice (as seen from examples in Desfontaines [2021]),

(ii) We also consider tiny values of σ in $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ to understand how the different generators behave in the regime interpolating between "no privacy" and "commonly used regimes of privacy".

We observe that for non-private SGD, Balls-and-Bins and Shuffling have similar utility and improve significantly over Poisson subsampling. For DP-SGD, we observe similar trends for noise multipliers at most 0.001, but for higher noise multipliers that could be deemed more relevant in practice, the different batch generators all have similar utility.

Next, we plot bounds on $\delta_{\mathcal{G}}(\varepsilon)$ for $\mathcal{G} \in \{\mathcal{S}, \mathcal{P}, \mathcal{B}\}$ for different combinations of σ and (expected) batch size in Figure 6. For $\delta_{\mathcal{P}}$, we plot both upper and lower bounds as computed using the `dp_accounting` library [Google's DP Library., 2020]. For $\delta_{\mathcal{S}}$ we plot our lower bound from Section 3.3. For $\delta_{\mathcal{B}}$, we plot a lower bound from (5.6), the mean of the Monte Carlo estimate (value q in Algorithm 8) and the upper confidence bound (value p in Algorithm 8) for error probability $\beta = 10^{-3}$. We find even the upper confidence bounds on $\delta_{\mathcal{B}}$ to be lower than $\delta_{\mathcal{P}}$ for the most part, with the exceptional cases when $\delta_{\mathcal{P}}$ is smaller than 10^{-7}, as this is the region where the concentration bounds are not strong enough. We believe that $\delta_{\mathcal{B}}(\varepsilon) < \delta_{\mathcal{P}}(\varepsilon)$ even in this regime.

7. Discussion

We uncover a substantial gap between the privacy guarantees of DP-SGD$_{\mathcal{P}}$ and DP-SGD$_{\mathcal{S}}$; the latter is often implemented in practice while the former is used for privacy accounting. To address this issue, we provide two possible solutions. First, we propose a modification of the Poisson batch generator where we perform truncation so that the batch sizes are bounded; we show that such a truncation allows for an efficient and scalable implementation.

Second, we introduce the Balls-and-Bins generator for DP-SGD, and show that it enjoys similar model utility as DP-SGD with shuffling, and enjoys privacy amplification that is similar to Poisson subsampling in practical regimes. In order to do so efficiently, we developed the techniques of importance sampling and order statistics sampling. While in our paper we primarily considered a single epoch of training, our approaches also extend to multiple epochs as discussed in Appendix G.4.

Our work leaves several directions for future investigation. The main open problem is to obtain a tight provable privacy accounting for ABLQ$_{\mathcal{S}}$ and ABLQ$_{\mathcal{B}}$. For the former, a concrete step would be to identify the tight dominating pair, i.e., by (dis)proving Conjecture 2.3. For the latter, the main limitation of our Monte Carlo approach is that it can only provide the high probability bounds; it would be interesting to obtain deterministic upper bounds that are accurate in relevant parameter regimes. An efficient method for tight accounting will also be useful to perform "inverse" accounting, namely to find σ for a desired choice of (ε, δ). Another alternative is to obtain RDP guarantees.

Subsequent to our work, Feldman and Shenfeld [2025] showed that the privacy guarantee of the balls-and-bins sampling is not worse than that of Poisson subsampling in a certain asymptotic sense. Furthermore, they also propose non-asymptotic bounds via decomposing

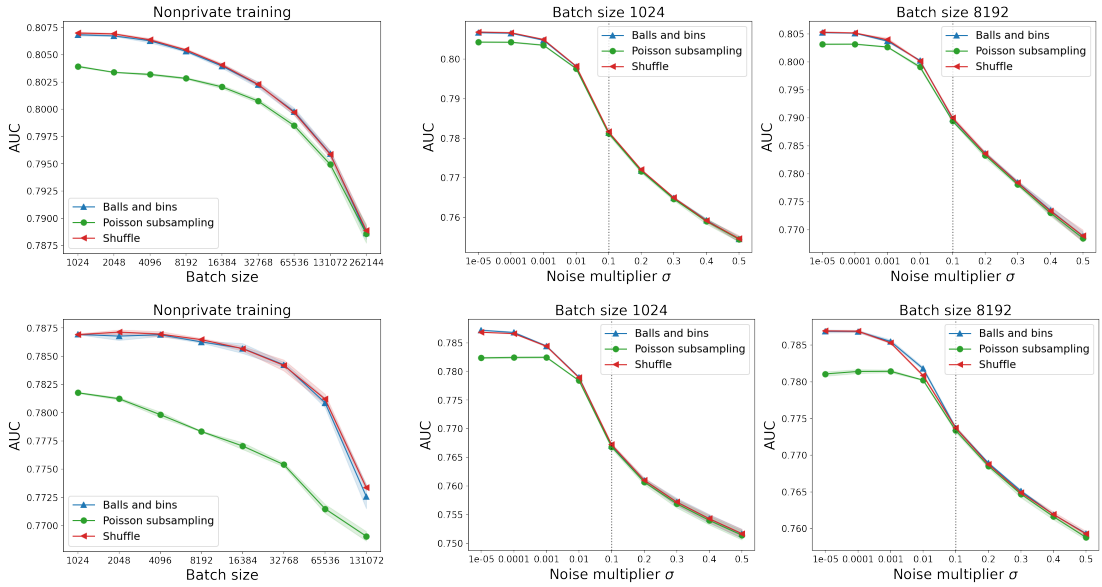

Figure 5: AUC values for one epoch of training with the Criteo Display Ads pCTR dataset (top) and the Criteo Sponsored Search Conversion Log dataset (bottom). On the left, we train without privacy and vary the batch size. In the middle and right, we train privately with varying σ, using (expected) batch sizes 1 024 (middle) and 8 192 (right). We use a log scale to the left of the vertical dotted line at $\sigma = 0.1$, and a linear scale to the right.

the privacy loss distribution of Poisson subsampling, as well as via RDP. These approaches are a promising avenue for overcoming the limitations of Monte Carlo sampling in this work.

Finally, we note that our privacy analysis for the truncated Poisson batch generator (Section 4.1) is likely not the optimal approach to account for the batch truncation; indeed, in a follow-up work, Ganesh [2025] has obtained a different (but not yet tight) privacy analysis that is tighter in some regimes of parameters. We do not optimize this further because we find that our approach already provides very minimal degradation to the choice of σ for a modest value of B relative to b. A more careful analysis could at best result in a slightly smaller B, which we do not consider as significant as the other questions discussed above.

Acknowledgments

We thank several anonymous reviewers of previous conference papers and of Journal of Privacy and Confidentiality for their thoughtful comments and suggestions that have improved the quality of this paper. We also thank the authors of Choquette-Choo et al. [2025] for helpful discussions regarding their concurrent work.

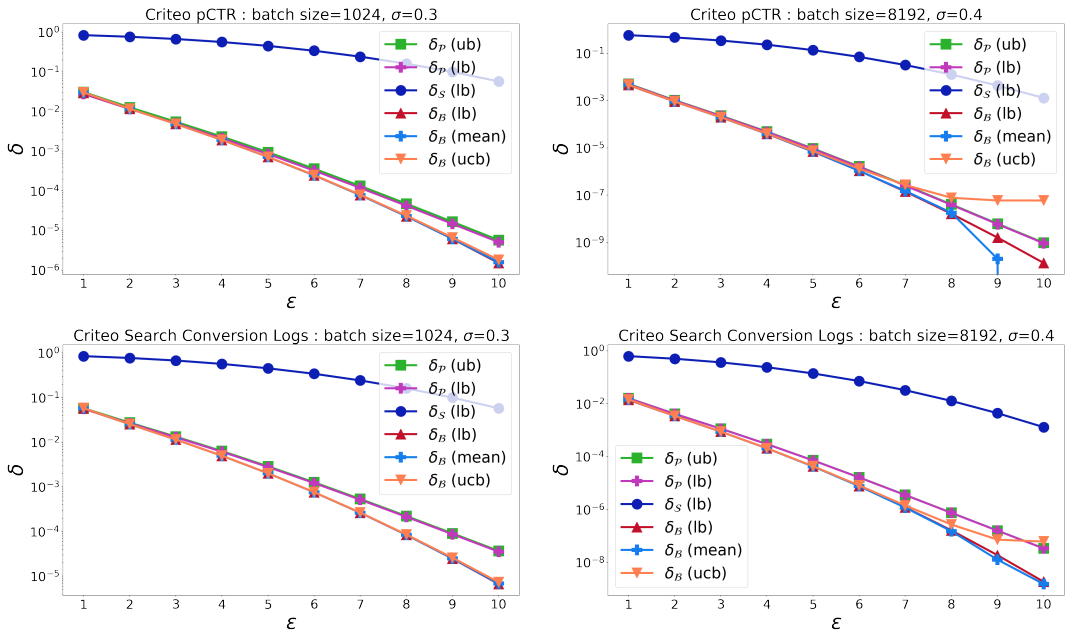

Figure 6: Bounds on $\delta_{\mathcal{P}}$, $\delta_{\mathcal{S}}$, and $\delta_{\mathcal{B}}$ are plotted for various values of ε for different (expected) batch size and σ. These mean and upper confidence bounds for $\delta_{\mathcal{B}}$ were obtained using order statistics sampling (specific orders and sample complexity specified in Appendix H).

References

M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *CCS*, pages 308–318, 2016. doi: 10.1145/2976749.2978318. URL https://dl.acm.org/doi/10.1145/2976749.2978318.

R. Anil, B. Ghazi, V. Gupta, R. Kumar, and P. Manurangsi. Large-scale differentially private BERT. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6481–6491, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.484. URL https://aclanthology.org/2022.findings-emnlp.484/.

B. Balle and Y.-X. Wang. Improving the Gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 394–403. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/balle18a.html.

B. Balle, L. Berrada, S. De, S. Ghalebikesabi, J. Hayes, A. Pappu, S. L. Smith, and R. Stanforth. JAX-Privacy: Algorithms for privacy-preserving machine learning in JAX, 2022. URL http://github.com/google-deepmind/jax_privacy.

S. Ben Slimane. Bounds on the distribution of a sum of independent lognormal random variables. *IEEE Trans. Comm.*, 49(6):975–978, 2001. doi: 10.1109/26.930627. URL https://doi.org/10.1109/26.930627.

C. A. Choquette-Choo, A. Ganesh, S. Haque, T. Steinke, and A. Thakurta. Near exact privacy amplification for matrix mechanisms. In *ICLR*, 2025. URL https://openreview.

net/forum?id=txV4dNeusx.

L. Chua, B. Ghazi, P. Kamath, R. Kumar, P. Manurangsi, A. Sinha, and C. Zhang. How private are DP-SGD implementations? In *ICML*, pages 8904–8918, 2024a. doi: 10.5555/3692070.3692425. URL https://dl.acm.org/doi/10.5555/3692070.3692425.

L. Chua, B. Ghazi, P. Kamath, R. Kumar, P. Manurangsi, A. Sinha, and C. Zhang. Scalable DP-SGD: Shuffling vs. Poisson subsampling. In *NeurIPS*, pages 70026–70047, 2024b. doi: 10.5555/3737916.3740154. URL https://dl.acm.org/doi/10.5555/3737916.3740154.

L. Chua, B. Ghazi, C. Harrison, P. Kamath, R. Kumar, E. J. Leeman, P. Manurangsi, A. Sinha, and C. Zhang. Balls-and-bins sampling for dp-sgd. In Y. Li, S. Mandt, S. Agrawal, and E. Khan, editors, *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pages 946–954. PMLR, 03–05 May 2025. URL https://proceedings.mlr.press/v258/chua25a.html.

H. A. David and H. N. Nagaraja. *Order Statistics*. John Wiley & Sons, 2004. doi: 10.1002/0471722162. URL https://onlinelibrary.wiley.com/doi/book/10.1002/0471722162.

S. De, L. Berrada, J. Hayes, S. L. Smith, and B. Balle. Unlocking high-accuracy differentially private image classification through scale. *arXiv:2204.13650*, 2022. URL https://arxiv.org/abs/2204.13650.

J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *C. ACM*, 51(1):107–113, 2008. doi: 10.1145/1327452.1327492. URL https://doi.org/10.1145/1327452.1327492.

D. Desfontaines. A list of real-world uses of differential privacy, Oct 2021. URL https://desfontain.es/blog/real-world-differential-privacy.html. Ted is writing things (personal blog).

T. Dockhorn, T. Cao, A. Vahdat, and K. Kreis. Differentially private diffusion models. *TMLR*, 2023. URL https://openreview.net/forum?id=ZPpQk7FJXF.

V. Doroshenko, B. Ghazi, P. Kamath, R. Kumar, and P. Manurangsi. Connect the dots: Tighter discrete approximations of privacy loss distributions. *PETS*, 2022(4):552–570, 2022. doi: 10.56553/popets-2022-0122. URL https://doi.org/10.56553/popets-2022-0122.

C. Dwork and G. N. Rothblum. Concentrated differential privacy. *arXiv:1603.01887*, 2016. URL https://arxiv.org/abs/1603.01887.

C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, pages 486–503, 2006a. doi: 10.1007/11761679_29. URL https://doi.org/10.1007/11761679_29.

C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, page 265–284, Berlin, Heidelberg, 2006b. Springer-Verlag. ISBN 3540327312. doi: 10.1007/11681878_14. URL https://doi.org/10.1007/11681878_14.

U. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. Amplification by shuffling: from local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, page 2468–2479, USA, 2019. Society for Industrial and Applied Mathematics. doi: 10.5555/3310435.3310586. URL https://dl.acm.org/doi/10.5555/3310435.3310586.

V. Feldman and M. Shenfeld. Privacy amplification by random allocation. In *NeurIPS*, 2025. URL https://neurips.cc/virtual/2025/loc/san-diego/poster/118441.

V. Feldman, A. McMillan, and K. Talwar. *Stronger Privacy Amplification by Shuffling for Renyi and Approximate Differential Privacy*, pages 4966–4981. doi: 10.1137/1.9781611977554.ch181. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977554.ch181.

V. Feldman, A. McMillan, and K. Talwar. Hiding among the clones: A simple and nearly optimal analysis of privacy amplification by shuffling. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 954–964, 2022. doi: 10.1109/FOCS52979.2021.00096. URL https://ieeexplore.ieee.org/document/9719772.

A. Ganesh. Tighter privacy analysis for truncated Poisson sampling. *arXiv:2508.15089*, 2025. URL https://arxiv.org/abs/2508.15089.

B. Ghazi, P. Kamath, R. Kumar, and P. Manurangsi. Faster privacy accounting via evolving discretization. In *ICML*, pages 7470–7483, 2022. URL https://proceedings.mlr.press/v162/ghazi22a.html.

Google. TensorFlow Privacy, 2025. URL https://github.com/tensorflow/privacy.

Google Cloud. Google Cloud Dataflow, 2025. URL https://cloud.google.com/dataflow.

Google's DP Library. DP Accounting Library, 2020. URL https://github.com/google/differential-privacy/tree/main/python/dp_accounting.

S. Gopi, Y. T. Lee, and L. Wutschitz. Numerical composition of differential privacy. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NeurIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393. doi: 10.5555/3540261.3541150. URL https://dl.acm.org/doi/10.5555/3540261.3541150.

J. He, X. Li, D. Yu, H. Zhang, J. Kulkarni, Y. T. Lee, A. Backurs, N. Yu, and J. Bian. Exploring the limits of differentially private deep learning with group-wise clipping. In *ICLR*, 2023. URL https://openreview.net/forum?id=oze0clVGPeX.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. ASA*, 58 (301):13–30, 1963. doi: 10.1080/01621459.1963.10500830. URL https://doi.org/10.1080/01621459.1963.10500830.

T. Igamberdiev, D. N. L. Vu, F. Kuennecke, Z. Yu, J. Holmer, and I. Habernal. DP-NMT: Scalable differentially private machine translation. In N. Aletras and O. De Clercq, editors, *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 94–105, St. Julians, Malta, Mar. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.eacl-demo.11. URL https://aclanthology.org/2024.eacl-demo.11/.

O. C. Jean-Baptiste Tien, joycenv. Display advertising challenge, 2014. URL https://kaggle.com/competitions/criteo-display-ad-challenge.

P. Kairouz, S. Oh, and P. Viswanath. The composition theorem for differential privacy. In *ICML*, pages 1376–1385, 2015. doi: 10.5555/3045118.3045265. URL https://dl.acm.org/doi/10.5555/3045118.3045265.

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*, volume 139 of *Proceedings of Machine Learning Research*, pages 5213–5225. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/kairouz21b.html.

A. Koskela, J. Jälkö, and A. Honkela. Computing tight differential privacy guarantees using FFT. In *AISTATS*, pages 2560–2569, 2020. URL https://proceedings.mlr.press/v108/koskela20b.html.

A. Koskela, M. A. Heikkilä, and A. Honkela. Numerical accounting in the shuffle model of differential privacy. *TMLR*, 2023, 2023. URL https://openreview.net/forum?id=11osftjEbF.

C. J. Lebeda, M. Regehr, and G. Kamath. Avoiding pitfalls for privacy accounting of subsampled mechanisms under composition. In *SaTML*, 2024. doi: 10.1109/SaTML64287.2025.00060. URL https://doi.org/10.1109/SaTML64287.2025.00060.

G. Marsaglia and W. W. Tsang. A simple method for generating gamma variables. *ACM Trans. Math. Softw.*, 26(3):363–372, 2000. doi: 10.1145/358407.358414. URL https://doi.org/10.1145/358407.358414.

S. Meiser and E. Mohammadi. Tight on budget? tight bounds for r-fold approximate differential privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 247–264, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356930. doi: 10.1145/3243734.3243765. URL https://doi.org/10.1145/3243734.3243765.

Microsoft. A fast algorithm to optimally compose privacy guarantees of differentially private (DP) mechanisms to arbitrary accuracy., 2021. URL https://github.com/microsoft/prv_accountant.

I. Mironov. Rényi Differential Privacy . In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275, Los Alamitos, CA, USA, Aug. 2017. IEEE Computer Society. doi: 10.1109/CSF.2017.11. URL https://doi.ieeecomputersociety.org/10.1109/CSF.2017.11.

N. Ponomareva, H. Hazimeh, A. Kurakin, Z. Xu, C. Denison, H. B. McMahan, S. Vassilvitskii, S. Chien, and A. G. Thakurta. How to dp-fy ML: A practical guide to machine learning with differential privacy. *J. AIR*, 77:1113–1201, 2023. doi: 10.1145/3580305.3599561. URL https://dl.acm.org/doi/10.1145/3580305.3599561.

L. Prediger and A. Koskela. Code for computing tight guarantees for differential privacy., 2020. URL https://github.com/DPBayes/PLD-Accountant.

D. M. Sommer, S. Meiser, and E. Mohammadi. Privacy loss classes: The central limit theorem in differential privacy. *PETS*, 2019(2):245–269, 2019. doi: 10.2478/popets-2019-0029. URL https://doi.org/10.2478/popets-2019-0029.

M. Tallis and P. Yadav. Reacting to Variations in Product Demand: An Application for Conversion Rate (CR) Prediction in Sponsored Search . In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1856–1864, Los Alamitos, CA, USA, Dec. 2018. IEEE Computer Society. doi: 10.1109/BigData.2018.8622223. URL https://doi.ieeecomputersociety.org/10.1109/BigData.2018.8622223.

X. Tang, A. Panda, M. Nasr, S. Mahloujifar, and P. Mittal. Private fine-tuning of large language models with zeroth-order optimization. *TMLR*, 2025. URL https://arxiv.org/abs/2401.04343.

TensorFlow Privacy, 2024. URL https://www.tensorflow.org/responsible_ai/privacy/api_docs/python/tf_privacy/compute_dp_sgd_privacy_statement. Note about compute_dp_sgd_privacy_statement.

The Apache Software Foundation. Apache Beam, 2025a. URL https://beam.apache.org.

The Apache Software Foundation. Apache Flink, 2025b. URL https://flink.apache.org.

The Apache Software Foundation. Apache Spark, 2025c. URL https://spark.apache.org.

S. Vadhan. *The Complexity of Differential Privacy*. Springer, 2017. doi: 10.1007/978-3-319-57048-8_7. URL https://doi.org/10.1007/978-3-319-57048-8_7.

J. T. Wang, S. Mahloujifar, T. Wu, R. Jia, and P. Mittal. A randomized approach to tight privacy accounting. In *NeurIPS*, pages 33856–33893, 2023. doi: 10.5555/3666122.3667591. URL https://dl.acm.org/doi/10.5555/3666122.3667591.

A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, and I. Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv:2109.12298*, 2021. URL https://arxiv.org/abs/2109.12298.

Y. Zhu, J. Dong, and Y. Wang. Optimal accounting of differential privacy via characteristic function. In *AISTATS*, pages 4782–4817, 2022. URL https://proceedings.mlr.press/v151/zhu22c.html.

Appendix A. Additional Comparisons Between $\mathcal{D}, \mathcal{P}, \mathcal{S}$

A.1. **ε vs. σ for Fixed δ and T.** We first plot ε against σ for fixed δ and T. We compute upper bounds on $\varepsilon_{\mathcal{P}}(\delta)$ using RDP as well as using PLD. These accounting methods are provided in the open source Google `dp_accounting` library [Google's DP Library., 2020].

In particular, we consistently find that for small values of σ, $\mathsf{ABLQ}_{\mathcal{S}}$ provides almost no improvement over $\mathsf{ABLQ}_{\mathcal{D}}$, and has $\varepsilon_{\mathcal{S}}$ that is significantly larger than $\varepsilon_{\mathcal{P}}$.

- In Figure 7, we set $\delta = 10^{-6}$ and number of steps $T = 100\,000$. In particular, for $\sigma = 0.4$, we find that $\varepsilon_{\mathcal{P}}(\delta) \leq 3$ (as per PLD accounting) and $\varepsilon_{\mathcal{P}}(\delta) \leq 4.71$ (as per RDP accounting), whereas on the other hand, $\varepsilon_{\mathcal{S}}(\delta) \geq 14.45$, which is very close to $\varepsilon_{\mathcal{D}}(\delta)$. For $\sigma = 1.3$, we find that $\varepsilon_{\mathcal{P}}(\delta) < 0.01$ (as per PLD accounting), whereas, $\varepsilon_{\mathcal{S}}(\delta) > 0.029$.

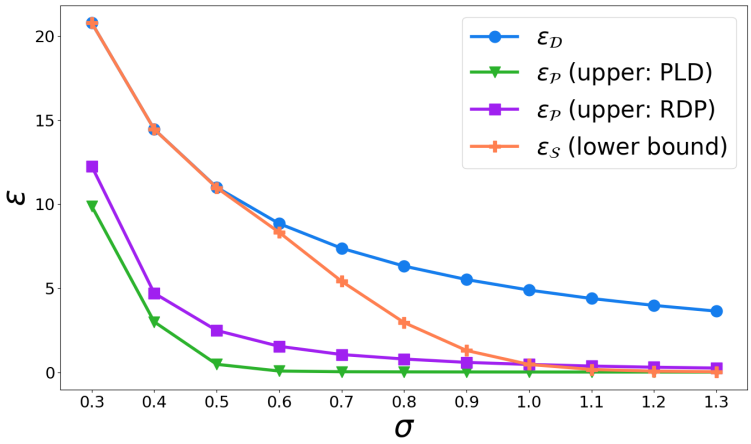

Figure 7: $\varepsilon_{\mathcal{D}}(\delta)$, upper bounds on $\varepsilon_{\mathcal{P}}(\delta)$ and a lower bound on $\varepsilon_{\mathcal{S}}(\delta)$ for varying σ and fixed $\delta = 10^{-6}$ and $T = 10\,000$.

- In Figure 8, we set $\delta = 10^{-5}$ and number of steps $T = 1\,000$. In particular, for $\sigma = 0.7$, $\varepsilon_{\mathcal{P}}(\delta) \leq 0.61$ (as per PLD accounting) and $\varepsilon_{\mathcal{P}}(\delta) \leq 1.64$ (as per RDP accounting), whereas on the other hand, $\varepsilon_{\mathcal{S}}(\delta) \geq 6.528$ and $\varepsilon_{\mathcal{D}}(\delta) \approx 6.652$. For $\sigma = 1.3$, we find that $\varepsilon_{\mathcal{P}}(\delta) < 0.092$ (as per PLD accounting), whereas, $\varepsilon_{\mathcal{S}}(\delta) > 0.83$.

A.2. **δ vs. ε for Fixed σ and T.** Next, we plot δ against ε for fixed σ and T. We compute upper bounds on $\delta_{\mathcal{P}}(\varepsilon)$ using RDP as well as using PLD.

In particular when σ is closer to 1.0, we find that our lower bound on $\delta_{\mathcal{S}}(\varepsilon)$ is distinctly smaller than $\delta_{\mathcal{D}}(\varepsilon)$, but still significantly larger than $\delta_{\mathcal{P}}(\varepsilon)$.

- In Figure 9, we set $\sigma = 0.8$ and number of steps $T = 1\,000$. In particular, while $\delta_{\mathcal{P}}(1) \leq 9.873 \cdot 10^{-9}$ (as per PLD accounting) and $\delta_{\mathcal{P}}(1) \leq 3.346 \cdot 10^{-5}$ (as per RDP accounting), we have that $\delta_{\mathcal{S}}(1) \geq 0.018$ and $\delta_{\mathcal{S}}(4) \geq 1.6 \cdot 10^{-4}$.

  For $\varepsilon = 4$, we find the upper bound using PLD accounting to be larger than the upper bound using RDP accounting. This is attributable to the numerical instability in PLD accounting when δ is very small.
- In Figure 10, we set $\sigma = 1.0$ and number of steps $T = 1\,000$. In particular, while $\delta_{\mathcal{P}}(1) \leq 2.06 \cdot 10^{-10}$ (as per PLD accounting) and $\delta_{\mathcal{P}}(1) \leq 8.45 \cdot 10^{-5}$ (as per RDP

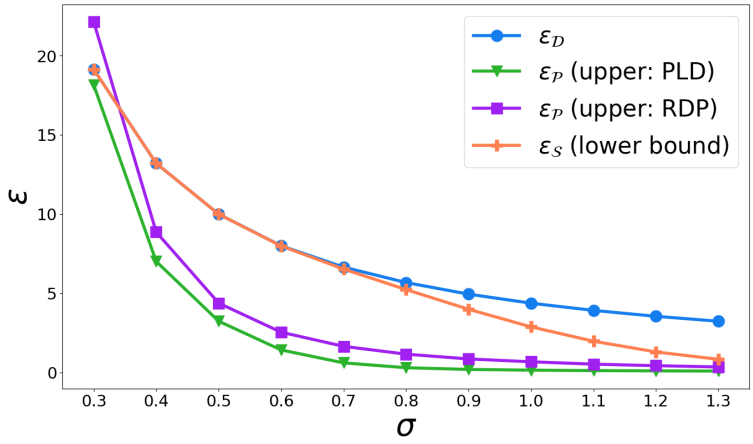

Figure 8: $\varepsilon_{\mathcal{D}}(\delta)$, upper bounds on $\varepsilon_{\mathcal{P}}(\delta)$ and a lower bound on $\varepsilon_{\mathcal{S}}(\delta)$ for varying σ and fixed $\delta = 10^{-5}$ and $T = 1\,000$.

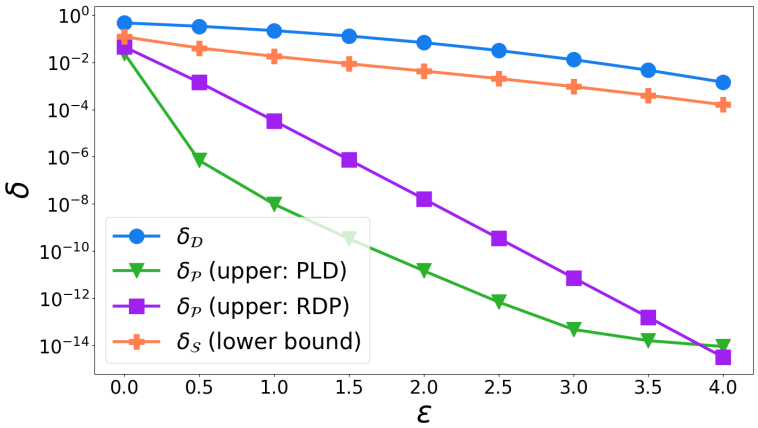

Figure 9: $\delta_{\mathcal{D}}(\varepsilon)$, upper bounds on $\delta_{\mathcal{P}}(\varepsilon)$ and a lower bound on $\delta_{\mathcal{S}}(\varepsilon)$ for varying ε and fixed $\sigma = 0.8$ and $T = 1\,000$.

accounting), we have that $\delta_{\mathcal{S}}(1) \geq 0.004$ and $\delta_{\mathcal{S}}(4) \geq 4.38 \cdot 10^{-7}$ (last one not shown in plot).

For $\varepsilon > 1.0$, we find the upper bound using PLD accounting appears to not decrease as much, which could be due to the numerical instability in PLD accounting when δ is very small.

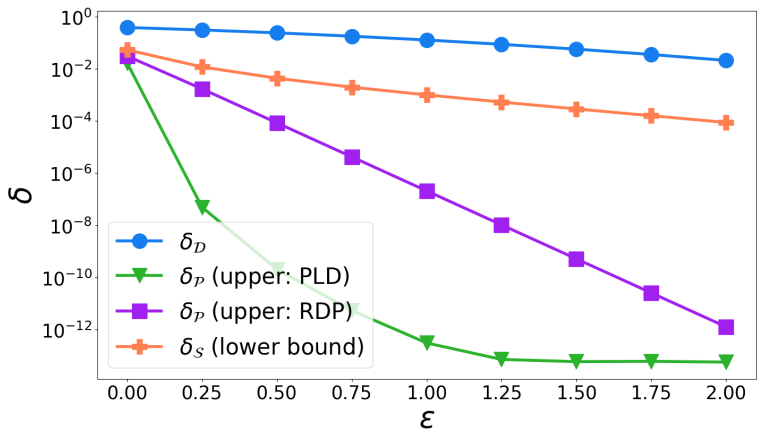

Figure 10: $\delta_{\mathcal{D}}(\varepsilon)$, upper bounds on $\delta_{\mathcal{P}}(\varepsilon)$ and a lower bound on $\delta_{\mathcal{S}}(\varepsilon)$ for varying ε and fixed $\sigma = 1.0$ and $T = 1\,000$.

Appendix B. Proof of Joint Convexity of Hockey Stick Divergence (Lemma 5.2)

Proof of Lemma 5.2. We have that

$$\begin{aligned} D_{e^\varepsilon}(P \parallel Q) &= \sup_E \{P(E) - e^\varepsilon Q(E)\} \\ &= \sup_E \left\{ \sum_{i=1}^m \alpha_i \left(P_i(E) - e^\varepsilon Q_i(E)\right) \right\} \\ &\le \sum_{i=1}^m \alpha_i \cdot \sup_E \{P_i(E) - e^\varepsilon Q_i(E)\} = \sum_{i=1}^m \alpha_i D_{e^\varepsilon}(P_i \parallel Q_i)\,. \end{aligned}$$

□

Appendix C. Proof of Theorem 3.1 ($\mathsf{ABLQ}_{\mathcal{D}}$ vs. $\mathsf{ABLQ}_{\mathcal{S}}$)

It follows from the joint convexity property of hockey stick divergence that shuffling the dataset first cannot degrade the privacy guarantee of *any* mechanism as shown below.

Lemma C.1. *Fix a mechanism $\mathcal{M} : \mathcal{X}^* \to \Delta_{\mathcal{O}}$, and let $\mathcal{M}_{\mathcal{S}}$ be defined as $\mathcal{M}_{\mathcal{S}}(\boldsymbol{x}) := \mathcal{M}(\boldsymbol{x}_\pi)$ for a random permutation π over $[n]$ where $\boldsymbol{x}_\pi := (x_{\pi(1)}, \ldots, x_{\pi(n)})$. Then, if $\mathcal{M}$ satisfies (ε, δ)-DP, then $\mathcal{M}_{\mathcal{S}}$ also satisfies (ε, δ)-DP.*

Proof. Consider any adjacent pair of dataset $\boldsymbol{x} \sim \boldsymbol{x}'$. For any permutation π over $[n]$, let $P_\pi := \mathcal{M}(\boldsymbol{x}_\pi)$ and $Q_\pi := \mathcal{M}(\boldsymbol{x}'_\pi)$, and let $P = \mathcal{M}_{\mathcal{S}}(\boldsymbol{x})$ and $Q = \mathcal{M}_{\mathcal{S}}(\boldsymbol{x}')$. It is easy to see that

$$P = \tfrac{1}{n!}\textstyle\sum_\pi P_\pi \qquad \text{and} \qquad Q = \tfrac{1}{n!}\textstyle\sum_\pi Q_\pi\,.$$

Since $\mathcal{M}$ satisfies (ε, δ)-DP it follows that $D_{e^\varepsilon}(P_\pi \parallel Q_\pi) \le \delta$ for all permutations π. Thus, from Lemma 5.2, it follows that $D_{e^\varepsilon}(P \parallel Q) \le \frac{1}{n!}\sum_\pi D_{e^\varepsilon}(P_\pi \parallel Q_\pi) \le \delta$. Hence $\mathcal{M}_{\mathcal{S}}$ also satisfies (ε, δ)-DP. □

Proof of Theorem 3.1. The proof follows by observing that if we choose $\mathcal{M} = \mathsf{ABLQ}_{\mathcal{D}}$ in Lemma C.1, then $\mathsf{ABLQ}_{\mathcal{S}}$ is precisely the corresponding mechanism $\mathcal{M}_{\mathcal{S}}$. □

Appendix D. Proof of Theorem 3.2 ($\mathsf{ABLQ}_{\mathcal{D}}$ vs. $\mathsf{ABLQ}_{\mathcal{P}}$)

We first state and prove some intermediate statements required for the proof of Theorem 3.2. We use the Gaussian measure of a halfspace.

Proposition D.1. *For $P = \mathcal{N}(\mu, \sigma^2 I)$ and the set $E := \{w \in \mathbb{R}^d : a^\top w - b \geq 0\}$, it holds that $P(E) = \Phi\left(\frac{a^\top \mu - b}{\sigma \|a\|_2}\right)$.*

Proof. This follows by observing that, if $w \sim \mathcal{N}(\mu, \sigma^2 I)$, then $a^\top w - b \sim \mathcal{N}(a^\top \mu - b, \sigma^2 \|a\|_2^2)$. □

Proposition D.2. *For all $T \in \mathbb{N}$ and distributions A, B, it holds that $D_1(A^{\otimes T} \parallel B^{\otimes T}) \leq 1 - (1 - D_1(A \parallel B))^T$. And hence $D_1(A^{\otimes T} \parallel B^{\otimes T}) \leq T \cdot D_1(A \parallel B)$ and equality holds only if $T = 1$ or $D_1(A \parallel B) = 0$.*

Proof. Recall that $D_1(A \parallel B)$ is the total variation distance between A and B, which has the following characterization $\inf_{(X,Y)} \Pr[X \neq Y]$ where (X, Y) is a coupling such that $X \sim A, Y \sim B$. Given any coupling (X, Y) for A, B, we construct a coupling $((X_1, \ldots, X_T), (Y_1, \ldots, Y_T))$ of $A^{\otimes T}, B^{\otimes T}$ by sampling (X_i, Y_i) independently according to the coupling (X, Y). From this, we have

$$\Pr[(X_1, \ldots, X_T) \neq (Y_1, \ldots, Y_T)] = 1 - \Pr[(X_1, \ldots, X_T) = (Y_1, \ldots, Y_T)] = 1 - \Pr[X = Y]^T.$$

By taking the infimum over all (X, Y) such that $X \sim A, Y \sim B$, the desired bound follows. □

We also note that a simple observation that for all P, Q, the hockey stick divergence $D_{e^\varepsilon}(P \parallel Q)$ is a 1-Lipschitz in e^ε.

Proposition D.3. *For $\varepsilon_1 < \varepsilon_2$, it holds that $D_{e^{\varepsilon_1}}(P \parallel Q) - D_{e^{\varepsilon_2}}(P \parallel Q) \leq e^{\varepsilon_2} - e^{\varepsilon_1}$.*

Proof. We have that

$$\begin{aligned} D_{e^{\varepsilon_1}}(P \parallel Q) - D_{e^{\varepsilon_2}}(P \parallel Q) &= \sup_E \{P(E) - e^{\varepsilon_1} Q(E)\} - \sup_{E'} \{P(E') - e^{\varepsilon_2} Q(E')\} \\ &\leq \sup_E \{P(E) - e^{\varepsilon_1} Q(E) - P(E) + e^{\varepsilon_2} Q(E)\} \\ &= (e^{\varepsilon_2} - e^{\varepsilon_1}) \cdot \sup_E Q(E) \\ &= e^{\varepsilon_2} - e^{\varepsilon_1}. \end{aligned}$$

□

Proof of Theorem 3.2. We prove each part as follows:

For part (a), first we consider the case of $\varepsilon = 0$. In this case, $D_{e^\varepsilon}(P \parallel Q)$ is simply the total variation distance between P and Q. Recall that $P_{\mathcal{P}} = A^{\otimes T}$ and $Q_{\mathcal{P}} = B^{\otimes T}$, where $A = (1 - \frac{1}{T})Q_{\mathcal{D}} + \frac{1}{T}P_{\mathcal{D}}$ and $B = Q_{\mathcal{D}}$. Observe that $D_1(A \parallel B) = \frac{1}{T} \cdot D_1(P_{\mathcal{D}} \parallel Q_{\mathcal{D}})$. Thus, we have that

$$D_1(P_{\mathcal{P}} \parallel Q_{\mathcal{P}}) = D_1(A^{\otimes T} \parallel B^{\otimes T}) < T \cdot D_1(A \parallel B) = D_1(P_{\mathcal{D}} \parallel Q_{\mathcal{D}}),$$

where the inequality follows from Proposition D.2. Note that the inequality is strict for $T > 1$. Since $D_{e^\varepsilon}(P \parallel Q)$ is continuous in ε (see Proposition D.3), there exists some $\varepsilon_0 > 0$, such that for all $\varepsilon \in [0, \varepsilon_0)$, $\delta_{\mathcal{D}}(\varepsilon) > \delta_{\mathcal{P}}(\varepsilon)$.

For part (b), we construct an explicit bad event $E \subseteq \mathbb{R}^T$ such that $P_{\mathcal{P}}(E) - e^\varepsilon Q_{\mathcal{P}}(E) > D_{e^\varepsilon}(P_{\mathcal{D}} \parallel Q_{\mathcal{D}})$. In particular, we consider:

$$E := \left\{ w \in \mathbb{R}^T \middle| \textstyle\sum_i w_i > (\varepsilon + \log 2 + T \log T)\sigma^2 + \frac{T}{2} \right\}.$$

The choice of E is such that,

$$\begin{aligned}
\log \tfrac{P_{\mathcal{P}}(w)}{Q_{\mathcal{P}}(w)} &= \textstyle\sum_{t=1}^T \log \frac{A(w_t)}{B(w_t)} \\
&= \textstyle\sum_{t=1}^T \log \left(1 - \frac{1}{T} + \frac{1}{T} \cdot e^{\frac{2w_t - 1}{2\sigma^2}}\right) \\
&\geq \textstyle\sum_{t=1}^T \left(\frac{2w_t - 1}{2\sigma^2} - \log T\right) \\
&\geq \tfrac{\sum_{t=1}^T w_t}{\sigma^2} - T \log T - \tfrac{T}{2\sigma^2} \\
&\geq \varepsilon + \log 2 \qquad \text{for all } w \in E.
\end{aligned}$$

Hence it follows that $\log \frac{P_{\mathcal{P}}(E)}{Q_{\mathcal{P}}(E)} \geq \varepsilon + \log 2$, or equivalently, $P_{\mathcal{P}}(E) \geq 2e^\varepsilon Q_{\mathcal{P}}(E)$. This implies that $D_{e^\varepsilon}(P_{\mathcal{P}} \parallel Q_{\mathcal{P}}) \geq \frac{1}{2}P_{\mathcal{P}}(E)$.

Next, we obtain a lower bound on $P_{\mathcal{P}}(E)$. For $N_\mu := \mathcal{N}(\mu, \sigma^2 I)$ and $p_k = \frac{1}{T^{-k}}(1 - \frac{1}{T})^{T-k}$, it holds that

$$P_{\mathcal{P}}(E) = \textstyle\sum_{\mu \in \{0,1\}^T} p_{\|\mu\|_1} N_\mu(E) \geq N_{\mathbf{0}}(E) = \Phi\left(-\tfrac{\varepsilon\sigma}{\sqrt{T}} - \tfrac{(T \log T + \log 2)\sigma}{\sqrt{T}} - \tfrac{\sqrt{T}}{2\sigma}\right), \quad \text{(D.1)}$$

where we use Proposition D.1 in the last two steps. On the other hand, we have from Proposition 2.1 that

$$\delta_{\mathcal{D}}(\varepsilon) = \Phi\left(-\varepsilon\sigma + \tfrac{1}{2\sigma}\right) - e^\varepsilon \Phi\left(-\varepsilon\sigma - \tfrac{1}{2\sigma}\right) \leq \Phi\left(-\varepsilon\sigma + \tfrac{1}{2\sigma}\right). \quad \text{(D.2)}$$

There exists a sufficiently large ε_1 such that

$$\begin{aligned}
\delta_{\mathcal{P}}(\varepsilon) &\geq D_{e^\varepsilon}(P_{\mathcal{P}} \parallel Q_{\mathcal{P}}) \\
&\geq \tfrac{1}{2}P_{\mathcal{P}}(E) \\
&\geq \tfrac{1}{2}\Phi\left(-\tfrac{\varepsilon\sigma}{\sqrt{T}} - \tfrac{(T \log T + \log 2)\sigma}{\sqrt{T}} - \tfrac{\sqrt{T}}{2\sigma}\right) && \text{(D.3)} \\
&\geq \Phi\left(-\varepsilon\sigma + \tfrac{1}{2\sigma}\right) \qquad (\text{for } \varepsilon > \varepsilon_1) && \text{(D.4)} \\
&\geq \delta_{\mathcal{D}}(\varepsilon),
\end{aligned}$$

by noting that for large ε the most significant term inside $\Phi(\cdot)$ in (D.3) is $-\varepsilon\sigma/\sqrt{T}$, whereas in (D.4) the most significant term inside $\Phi(\cdot)$ is $-\varepsilon\sigma$, which decreases much faster as $\varepsilon \to \infty$, for a fixed $T > 1$ and $\sigma > 0$. □

Appendix E. Massively Parallel Implementation of Truncated Poisson Subsampling

We use massively parallel computation to generate batches with truncated Poisson subsampling in a scalable manner. Given the input parameters b, B, T, and n, we first compute the maximum batch size B such that $\Psi(n, b, B) \cdot T \cdot (1 + e^{\varepsilon}) \leq 10^{-5} \cdot \delta$. For each example in the input dataset, we generate a list of batches that the example would be in when sampled using Poisson subsampling. While a naive implementation would sample T Bernoulli random variables with parameter b/n, this can be made efficient by sampling the indices of the batches containing the examples directly, since the difference between two consecutive such indices is distributed as a geometric random variable with parameter b/n. We then `group` the examples by the batches, and subsample each batch uniformly, without replacement, to obtain a batch of size at most B. For batches with size smaller than B, we pad the batch with examples such that every batch has size B. In order to differentiate the padding examples from the non-padding examples, we add a weight to all the examples, where the non-padding examples have weight 1 and the padding examples have weight 0. During the training, we use a weighted loss function using these weights, such that the padding examples do not have any effect on the training loss.

We include a code snippet for implementing truncated Poisson subsampling using using Apache `beam` [The Apache Software Foundation, 2025a] in Python, which can be implemented on distributed platforms such as The Apache Software Foundation [2025b], The Apache Software Foundation [2025c], Google Cloud [2025].

```python
import apache_beam as beam
import numpy as np
import tensorflow as tf

class PoissonSubsample(beam.PTransform):
  """Generate mini-batches of examples using poisson subsampling.

  Attributes:
    max_batch_size: Maximum mini-batch size.
    num_batches: Number of mini-batches.
    subsampling_probability: Probability of sampling each example in each mini-batch.
    sample_size: Number of samples to sample at a time.
  """

  def __init__(
      self,
      max_batch_size: int,
      num_batches: int,
      subsampling_probability: float,
      sample_size: int,
  ):
    self._max_batch_size = max_batch_size
    self._num_batches = num_batches
    self._subsampling_probability = subsampling_probability
    self._sample_size = sample_size
    self._rng = np.random.default_rng()

  def get_batch_indices(self):
    """Returns the indices of the mini-batches that an example is in.
```

```
    Assuming that an example is sampled in each mini-batch using poisson subsampling,
    return the list of indices of the mini-batches that the example is in.
    """
    largest_batch_index = 0
    batch_indices = np.array([])
    if self._subsampling_probability == 0.0:
      return batch_indices
    while largest_batch_index < self._num_batches:
      # Sample batches using geometric distribution
      geometric_samples = self._rng.geometric(
          p=self._subsampling_probability, size=self._sample_size
      )
      batch_indices = np.concatenate(
          (batch_indices, largest_batch_index + np.cumsum(geometric_samples))
      )
      largest_batch_index = batch_indices[-1]
    return batch_indices[batch_indices <= self._num_batches]

  def _add_padding(self, batch):
    """Returns mini-batch padded to max_batch_size with padding examples.

    Pads input mini-batch to size max_batch_size by adding padding examples. The
    padding examples are specified by adding a weight to all the examples, where
    padding examples have weight 0 and non-padding examples have weight 1.

    Args:
      batch: A tuple of (batch_id, list of examples)
    """
    batch_id, examples = batch
    examples = list(examples)
    if len(examples) < self._max_batch_size:
      padding_example = tf.train.Example()
      padding_example.CopyFrom(examples[0])
      padding_example.features.feature['weight'].float_list.value[:] = [0.0]
      examples.extend(
          [padding_example] * (self._max_batch_size - len(examples))
      )
    return (batch_id, examples)

  def expand(self, pcoll):
    def generate_batch_ids(example):
      # Convert to pairs consisting of (batch id, example)
      for batch_id in self.get_batch_indices():
        yield (int(batch_id), example)

    def _add_weights(example):
      # Add weight with value 1 to indicate non-padding examples
      weighted_example = tf.train.Example()
      weighted_example.CopyFrom(example)
      weighted_example.features.feature['weight'].float_list.value[:] = [1.0]
      return weighted_example

    # Group elements into batches keyed by the batch id
    grouped_pcoll = (
        pcoll
        | 'Add weights' >> beam.Map(_add_weights)
        | 'Key by batch id' >> beam.FlatMap(generate_batch_ids)
        | 'Sample up to max_batch_size elements per batch'
        >> beam.combiners.Sample.FixedSizePerKey(self._max_batch_size)
        | 'Add padding' >> beam.Map(self._add_padding)
```

```
)
return grouped_pcoll
```

Appendix F. Incomparability of Dominating Pairs for $\mathsf{ABLQ}_{\mathcal{B}}$ and $\mathsf{ABLQ}_{\mathcal{P}}$

We elaborate on Remark 5.5 showing that $\mathsf{ABLQ}_{\mathcal{B}}$ and $\mathsf{ABLQ}_{\mathcal{P}}$ have incomparable privacy guarantees.

Theorem F.1. *For all $\sigma > 0$ and $T > 1$, there exists $\varepsilon_0, \varepsilon_1 \in \mathbb{R}$ such that*
(a) $D_{e^{\varepsilon_0}}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}}) > D_{e^{\varepsilon_0}}(P_{\mathcal{P}} \parallel Q_{\mathcal{P}})$, *and*
(b) $D_{e^{\varepsilon_1}}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}}) < D_{e^{\varepsilon_1}}(P_{\mathcal{P}} \parallel Q_{\mathcal{P}})$.

We use the following lemma regarding KL divergence, defined for probability distributions P, Q over the space Ω as $\mathsf{KL}(P \parallel Q) := \int_{\Omega} \log \frac{dP}{dQ}(\omega) \cdot dP(\omega)$.

Lemma F.2. *Let P be a joint distribution over $\Omega := \Omega_1 \times \cdots \times \Omega_n$, and let $Q = Q_1 \otimes \cdots \otimes Q_T$ be a product distribution over Ω. Then, for $P_1, \ldots, P_T$ being the marginal distributions of P over $\Omega_1, \ldots, \Omega_T$ respectively, it holds that*

$$\mathsf{KL}(P \parallel Q) \geq \mathsf{KL}(P_1 \otimes \cdots \otimes P_T \parallel Q)\,.$$

Moreover, equality holds if and only if P is a product distribution.

We omit a full proof of Lemma F.2. Nonetheless it follows from a simple observation that for a product distribution Q, it holds that

$$\mathsf{KL}(P \parallel Q) = \mathsf{KL}(P \| P_1 \otimes \cdots \otimes P_T) + \mathsf{KL}(P_1 \otimes \cdots \otimes P_T \parallel Q)$$

and the first term on the right hand side is non-negative and equals zero if and only if P is a product distribution.

Fact F.3 (Post-processing inequality for KL-divergence)**.** *For distributions P, Q over Ω, and distributions A, B over Γ, if there exists $f : \Omega \to \Gamma$ such that $f(P) = A$ and $f(Q) = B$, then $\mathsf{KL}(P \parallel Q) \geq \mathsf{KL}(A \parallel B)$.*

Lemma F.4 (Converse to Lemma 1.3; [Kairouz et al., 2015, Theorem 2.5])**.** *For distributions P, Q over Ω, and distributions A, B over Γ, if $(P, Q) \succcurlyeq (A, B)$ then there exists $f : \Omega \to \Gamma$ such that simultaneously $f(P) = A$ and $f(Q) = B$.*

Proof of Theorem F.1. Part (a) follows from the observation that $Q_{\mathcal{P}} = Q_{\mathcal{B}}$ and $P_{\mathcal{P}}$ can be obtained as simply the product of the marginal distributions of $P_{\mathcal{B}}$. Thus, applying Lemma F.2, we get that $\mathsf{KL}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}}) > \mathsf{KL}(P_{\mathcal{P}} \parallel Q_{\mathcal{P}})$, with strict inequality because $P_{\mathcal{B}}$ is not a product distribution. Thus, by the contrapositive of Fact F.3, we get that there does not exist a post-processing that simultaneously maps $P_{\mathcal{P}}$ to $P_{\mathcal{B}}$ and $Q_{\mathcal{P}}$ to $Q_{\mathcal{B}}$. Finally, by Lemma 1.3, we conclude that $(P_{\mathcal{P}}, Q_{\mathcal{P}}) \not\succcurlyeq (P_{\mathcal{B}}, Q_{\mathcal{B}})$ or in other words, there exists an $\varepsilon_0 \in \mathbb{R}$ such that $D_{e^{\varepsilon_0}}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}}) > D_{e^{\varepsilon_0}}(P_{\mathcal{P}} \parallel Q_{\mathcal{P}})$.

Part (b) follows immediately from Theorem 5.4. □

While Theorem 5.4 gives us that there exists an $\varepsilon \geq 0$ such that $\delta_{\mathcal{B}}(\varepsilon) < \delta_{\mathcal{P}}(\varepsilon)$ for all $\sigma > 0$ and $T > 1$ (in fact, this holds for sufficiently large ε), interestingly Theorem F.1 does *not* imply that there exists $\varepsilon \geq 0$ such that $\delta_{\mathcal{B}}(\varepsilon) > \delta_{\mathcal{P}}(\varepsilon)$, since $\delta_{\mathcal{B}}(\varepsilon)$ corresponds to $\max\{D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}}), D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})\}$. Whether there always exists such an $\varepsilon \geq 0$ for all $\sigma > 0$ and $T > 1$ is left open for future investigation.

Appendix G. Importance and Order Statistics Sampling

We describe how to efficiently perform importance sampling as described in Algorithm 9 for the pair $(P_{\mathcal{B}}, Q_{\mathcal{B}})$ as well as the proof that Algorithm 10 samples from the joint distribution of order statistics. In order to do so, we use the connection between the Beta distribution and order statistics [see, e.g., David and Nagaraja, 2004].

First, we establish some notation that we use throughout this section. Let $\mathsf{Unif}[a, b]$ denote the uniform distribution over the interval $[a, b]$. For any distribution P over $\mathbb{R}$, let $\mathsf{CDF}_P(x) := \Pr_{z\sim P}[z \leq x]$ denote the cumulative density function, and let $\mathsf{CDF}_P^{-1}(\cdot)$ denote its inverse.[9] For any event (measurable set) E, let $P|_E$ denote the distribution of P conditioned on event E. In this work, we only use distributions with probability measures that are continuous with respect to the Lebesgue measure. Even though the following techniques extend to the non-continuous distributions, we assume that distributions are continuous below.

Definition G.1 (Beta Distribution)**.** The $\mathsf{Beta}(\alpha, \beta)$ distribution over $[0, 1]$ is defined by the density function

$$f(x; \alpha, \beta) := \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1 - x)^{\beta-1}.$$

Fact G.2 (Order Statistics and Beta Distribution)**.** *The random variable $y^{(k)}$ that is the kth largest element among $x_1, \ldots, x_R \sim \mathsf{Unif}[0, 1]$ is distributed as $\mathsf{Beta}(R - k + 1, k)$.*

An important primitive we use in our sampling methods is the ability to efficiently sample from $\mathsf{Beta}(\alpha, \beta)$ distributions [see, e.g., Marsaglia and Tsang, 2000], with efficient implementations available, for example in Python, using the class `scipy.stats.beta`.

Fact G.3 (Probability Integral Transform)**.** *Let P be any distribution over $\mathbb{R}$. For $x \sim P$, $\mathsf{CDF}_P(x)$ is distributed as $\mathsf{Unif}[0, 1]$. Conversely, for $y \sim \mathsf{Unif}[0, 1]$, $\mathsf{CDF}_P^{-1}(y)$ is distributed as P.*

Furthermore it follows that, for any interval $[a, b] \in \mathbb{R}$, the distribution of $\mathsf{CDF}_P(x)$ for $x \sim P|_{[a,b]}$ is $\mathsf{Unif}[\mathsf{CDF}_P(a), \mathsf{CDF}_P(b)]$, and conversely for $y \sim \mathsf{Unif}[\mathsf{CDF}_P(a), \mathsf{CDF}_P(b)]$, $\mathsf{CDF}_P^{-1}(y)$ is distributed as $P|_{[a,b]}$.

Thus, Fact G.3 implies that for any distribution P over $\mathbb{R}$ for which CDF_P and CDF_P^{-1} are efficiently computable, it is possible to sample from P conditioned on the sample being in any specified range $[a, b]$. Since $\mathsf{CDF}_{\mathsf{Beta}(\alpha,\beta)}$, $\mathsf{CDF}_{\mathcal{N}(0,\sigma^2)}$ and their inverses are efficiently computable, for example in Python using the classes `scipy.stats.beta` and `scipy.stats.norm` respectively, we can sample from the conditional $\mathsf{Beta}(\alpha, \beta)$ and $\mathcal{N}(0, \sigma^2)$ distributions.

[9] In cases where $\mathsf{CDF}_P(\cdot)$ is not a continuous function, the inverse is defined as $\mathsf{CDF}_P^{-1}(y) := \min_{x\in\mathbb{R}:\mathsf{CDF}_P(x)\geq y} x$; the minimum always exists since CDF_P is right continuous. However, since we only deal with distributions with continuous CDFs, this detail is not going to be important.

Fact G.2 and G.3 together suggest the following approach to sample a single order statistics for sampling R i.i.d. samples from P or $P|_{[a,b]}$.

Proposition G.4. *Let P be any distribution over $\mathbb{R}$. The random variable $y^{(k)}$ that is the kth largest element among $x_1, \dots, x_R \sim P$, $\mathsf{CDF}_P(y^{(k)})$ is distributed as $\mathsf{Beta}(R-k+1, k)$. Conversely, for $z \sim \mathsf{Beta}(R-k+1, k)$, $\mathsf{CDF}_P^{-1}(z)$ has the same distribution as $y^{(k)}$.*

Furthermore it follows that, for any interval $[a, b] \in \mathbb{R}$, the distribution of $\mathsf{CDF}_P(y^{(k)})$ for $y^{(k)}$ being the kth largest element among $x_1, \dots, x_R \sim P|_{[a,b]}$ is distributed as $\mathsf{CDF}_P(a) + (\mathsf{CDF}_P(b) - \mathsf{CDF}_P(a)) \cdot z$ for $z \sim \mathsf{Beta}(R-k+1, k)$, and conversely for $z \sim \mathsf{Beta}(R-k+1, k)$, $\mathsf{CDF}_P^{-1}(\mathsf{CDF}_P(a) + (\mathsf{CDF}_P(b) - \mathsf{CDF}_P(a)) \cdot z)$ has the same distribution as $y^{(k)}$.

G.1. **Efficient Importance Sampling.** In this section we describe how to efficiently estimate $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$ and $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$ using Algorithm 9. We use $\Phi_\sigma(\cdot)$ to denote $\mathsf{CDF}_{\mathcal{N}(0,\sigma^2)}$ for short.

G.1.1. *Estimating $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$.* In this case, we wish to estimate $\mathbb{E}_{x \sim Q_{\mathcal{B}}|_{E_\varepsilon}} \max\{0, 1 - e^{\varepsilon - L_{Q_{\mathcal{B}} \parallel P_{\mathcal{B}}}(x)}\}$ where $Q_{\mathcal{B}} = \mathcal{N}(0, \sigma^2 I_T)$ and $E_\varepsilon := \{x :\in \mathbb{R}^T : \max_{t \in [T]} x_t \le C_\varepsilon\}$ for $C_\varepsilon := \frac{1}{2} + \sigma^2 \cdot (\log T - \varepsilon)$. In order to sample from $Q_{\mathcal{B}}|_{E_\varepsilon}$, we observe that this is equivalent to sampling T coordinates i.i.d. from $\mathcal{N}(0, \sigma^2)|_{\{x : x \le C_\varepsilon\}}$. This can be done using Fact G.3, by sampling $y_t \sim \mathsf{Unif}[0, \Phi_\sigma(C_\varepsilon)]$ and returning $x_t = \Phi_\sigma^{-1}(y_t)$ for each $t \in [T]$.

G.1.2. *Estimating $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$.* In this case, we wish to estimate $\mathbb{E}_{x \sim P_1|_{E_\varepsilon}} \max\{0, 1 - e^{\varepsilon - L_{P_{\mathcal{B}} \parallel Q_{\mathcal{B}}}(x)}\}$ where $P_1 = \mathcal{N}(e_1, \sigma^2 I_T)$ and $E_\varepsilon := \{x : \max\{x_1 - 1, \max_{t>1} x_t\} \ge C_\varepsilon\}$ for $C_\varepsilon = \frac{1}{2} + \sigma^2 \cdot \left(\varepsilon - \log\left(1 + \frac{e^{1/\sigma^2} - 1}{T}\right)\right)$. The choice of E_ε is such that for $x \sim P_1|_{E_\varepsilon}$, the distribution of $x - e_1$ is the same as $\mathcal{N}(0, \sigma^2 I_T)|_{\{x : \max_t x_t \ge C_\varepsilon\}}$. In Algorithm 11, we provide a generic algorithm that for any distribution P over $\mathbb{R}$, samples from the distribution $P^{\otimes T}|_{\{x : \max_t x_t \ge C\}}$, i.e., samples from T i.i.d. samples from P conditioned on the maximum value being at least C. Thus, we can sample from $P_1|_{E_\varepsilon}$ by sampling $x' \sim \mathcal{N}(0, \sigma^2 I_T)|_{\{x' : \max_t x'_t \ge C_\varepsilon\}}$ using Algorithm 11, and returning $x = x' + e_1$.

Numerical Evaluation. To demonstrate the usefulness of our importance sampling method, in Figure 11, we plot the upper confidence bound on $\delta_{\mathcal{B}}(\varepsilon)$ as obtained via Algorithm 8 (i.e., without importance sampling) and via Algorithm 9 (i.e., with importance sampling) along the lower bound obtained via (5.6). The upper confidence bounds are obtained for error probability $\beta = 10^{-3}$. For a similar running time, we see that Algorithm 9 is able to get significantly tighter upper confidence bounds in each setting. This is made possible because the importance sampling is able to "zoom in" into events of tiny probability. For example, in the left part of Figure 11 for $T = 5\,000$ and $\sigma = 0.4$, at $\varepsilon = 9$, the importance sampler using $m = 200\,000$ samples is considering an event E_ε such that $P_{\mathcal{B}}(E_\varepsilon) \approx 3.75 \cdot 10^{-3}$, and on the right for $T = 10\,000$ and $\sigma = 0.35$, at $\varepsilon = 12$, the importance sampler using $m = 100\,000$ samples is considering an event E_ε such that $P_{\mathcal{B}}(E_\varepsilon) \approx 1.66 \cdot 10^{-4}$. Recall that the reduction in sample complexity due to our use of importance sampling is by a factor of $1/P_{\mathcal{B}}(E_\varepsilon)$.

Algorithm 11: Sampling from $P^{\otimes T}$, conditioned on the maximum value being at least C.

Input: Distribution P over $\mathbb{R}$, lower bound $C \in \mathbb{R}$ on the maximum value.
Output: Sample $x \sim P|_{\max_t x_t \geq C}$
$y_* \sim \mathsf{Beta}(T,1)|_{[\mathsf{CDF}_P(C),1]}$ (using Fact G.3)
$t_* \sim$ uniformly random coordinate in $[T]$
for $t \in \{1,\dots,T\}$ **do**
  if $t = t_*$ **then**
    $z_t \leftarrow y_*$
  end
  else
    $z_t \sim \mathsf{Unif}[0, y_*]$
  end
end
return $(\mathsf{CDF}_P^{-1}(z_t) : t \in [T])$

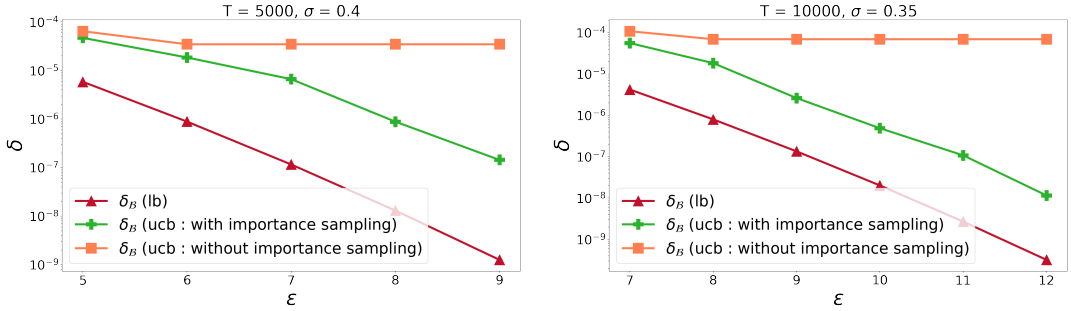

Figure 11: Upper confidence bounds on $\delta_{\mathcal{B}}(\varepsilon)$ against various values of ε for two settings of T and σ, with and without importance sampling. Additionally, lower bounds on $\delta_{\mathcal{B}}(\varepsilon)$ are included.

G.2. **Order Statistics Sampling.** We show that Algorithm 10 indeed samples from the joint distribution of order statistics of P.

Theorem G.5. *For any distribution P over $\mathbb{R}$, and number of random variables R and order statistic indices $k_1,\dots,k_r$, the values $(y^{(k_1)},\dots,y^{(k_r)})$ returned by Algorithm 10 are distributed as the $k_1,\dots,k_r$ largest elements among $x_1,\dots,x_R \sim P$.*

Proof. We prove the statement via induction on r. When $r = 1$, we have from Fact G.2, that $\mathsf{CDF}_P^{-1}(z_1)$ for $z_1 \sim \mathsf{Beta}(R-k_1+1,k_1)$ has the same distribution as the k_1th order statistic.

For $r > 1$, suppose we inductively assume that $(y^{(k_1)},\dots,y^{(k_{r-1})})$ are jointly distributed as the $(k_1,\dots,k_{r-1})$ order statistics. Note that $\mathsf{CDF}_P(y^{(k_i)}) = \prod_{j=1}^{i} z_j$ for all i. The conditional distribution of $y^{(k_r)}$ given $(y^{(k_1)},\dots,y^{(k_{r-1})})$ is the same as the $(k_r - k_{r-1})$th order statistic among $R - k_r$ random variables drawn from $P_{(-\infty,y^{(k_{r-1})}]}$. Using Proposition G.4, we have that for $z_r \sim \mathsf{Beta}(R-k_{r-1}+1, k_r-k_{r-1})$, $\mathsf{CDF}_P^{-1}(\mathsf{CDF}_P(y^{(k_{r-1})}) \cdot z_r))$ is distributed as per this conditional distribution. Since $\mathsf{CDF}_P(y^{(k_{r-1})}) = \prod_{j=1}^{r-1} z_j$ the induction argument is complete. □

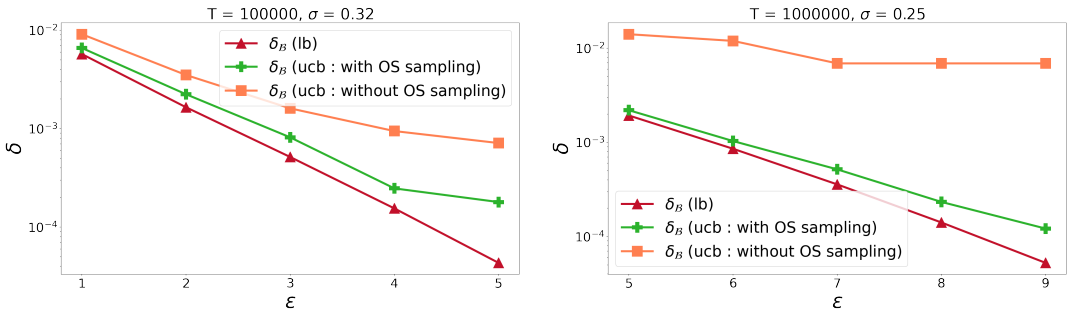

Figure 12: Upper confidence bounds on $\delta_{\mathcal{B}}(\varepsilon)$ against various values of ε for two settings of T and σ, with and without order statistics sampling for roughly the same running time complexity. Since order statistics sampling offers a significant speed up, it affords a larger sample complexity. Additionally, lower bounds on $\delta_{\mathcal{B}}(\varepsilon)$ are included.

Numerical evaluation. To demonstrate the usefulness of our order statistics sampling method, in Figure 12, we plot the upper confidence bound on $\delta_{\mathcal{B}}(\varepsilon)$ as obtained via Algorithm 8 as is (i.e., without order statistics sampling) and with order statistics sampling Algorithm 10 (i.e., with an upper bound on the loss function) along the lower bound obtained via (5.6). The sub-figures in Figure 12 were generated as follows.

- The figure on the left used $\sigma = 0.32$ and number of steps $T = 100\,000$. The estimates without order statistics used $m = 10\,000$ samples, whereas, the estimates with order statistics used $m = 100\,000$ samples, using the order statistics of $(1, 2, \ldots, 400, 410, \ldots, 1\,000, 1\,100, \ldots, 10\,000, 11\,000, \ldots, 50\,000)$ (a total of 590 orders). Despite using 10 times more samples, the estimation with order statistics ran in ~ 66 seconds, which is $\approx 25\%$ of the time needed without order statistics sampling (~ 268 seconds).
- The figure on the right used $\sigma = 0.25$ and number of steps $T = 1\,000\,000$. The estimates without order statistics used $m = 1\,000$ samples, whereas, the estimates with order statistics used $m = 300\,000$ samples, using the order statistics of $(1, 2, \ldots, 300, 310, \ldots, 1\,000, 1\,100, \ldots, 10\,000, 11\,000, \ldots, 100\,000, 110\,000, \ldots, 500\,000)$ (a total of 590 orders). Despite using 300 times more samples, the estimation with order statistics ran in ~ 203 seconds, which is $\approx 82\%$ of the time needed without order statistics sampling (~ 245 seconds).

Running times can vary significantly depending on the computing platform; these figures are offered only as a rough guide. The running time scales linearly with sample size and the number of order statistics and thus, these times are indicative of performance with more samples or varied order statistics.

G.3. **Combining Importance and Order Statistics Sampling.** We sketch how the techniques of importance sampling and order statistics sampling can be used together.

G.3.1. *Estimating* $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$. In this case, we wish to estimate $\mathbb{E}_{x \sim Q_{\mathcal{B}}|_{E_\varepsilon}} \max\{0, 1 - e^{\varepsilon - L_{Q_{\mathcal{B}} \parallel P_{\mathcal{B}}}(x)}\}$ where $Q_{\mathcal{B}} = \mathcal{N}(0, \sigma^2 I)$ and $E_\varepsilon := \{x :\in \mathbb{R}^T : \max_{t \in [T]} x_t \le C_\varepsilon\}$ for $C_\varepsilon := \frac{1}{2} + \sigma^2 \cdot (\log T - \varepsilon)$. We can sample the order statistics $y^{(k_1)}, \ldots, y^{(k_r)}$ for $x_1, \ldots, x_T \sim Q_{\mathcal{B}}|_{E_\varepsilon}$ by using a small variant of Algorithm 10 wherein we set $y^{(k_i)} \leftarrow \Phi_\sigma^{-1}(\Phi_\sigma(C_\varepsilon) \cdot \prod_{j=1}^{i} z_j)$, where the term $\Phi_\sigma(C_\varepsilon)$ essentially implements the conditioning on $x_1, \ldots, x_T \le C_\varepsilon$, via

Proposition G.4. Finally, we use Algorithm 9 where we replace $L_{Q_{\mathcal{B}}\|P_{\mathcal{B}}}(x)$ by an upper bound in terms of the order statistics given as,

$$\log T + \tfrac{1}{2\sigma^2} - \log\left(\textstyle\sum_{i=1}^{r}(k_i - k_{i-1}) \cdot e^{y^{(k_i)}/\sigma^2}\right).$$

G.3.2. *Estimating* $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$. In this case, we wish to estimate $\mathbb{E}_{x\sim P_1|_{E_\varepsilon}} \max\{0, 1 - e^{\varepsilon - L_{P_{\mathcal{B}}\|Q_{\mathcal{B}}}(x)}\}$ where $P_1 = \mathcal{N}(e_1, \sigma^2 I)$ and $E_\varepsilon := \{x : \max\{x_1 - 1, \max_{t>1} x_t\} \geq C_\varepsilon\}$ for $C_\varepsilon = \frac{1}{2} + \sigma^2 \cdot \left(\varepsilon - \log\left(1 + \frac{e^{1/\sigma^2}-1}{T}\right)\right)$. Recall that for $x \sim P_1|_{E_\varepsilon}$, the distribution of $x - e_1$ is the same as $\mathcal{N}(0, \sigma^2 I_T)|_{\{x\,:\,\max_t x_t \geq C_\varepsilon\}}$. We follow the first two steps of Algorithm 11 and sample $y_* \sim \mathsf{Beta}(T, 1)|_{[\Phi_\sigma(C_\varepsilon),1]}$, and sample t_* uniformly at random in $[T]$. There are two cases to handle:

- If $t_* = 1$, then we set $x_1 = y_* + 1$ and use Algorithm 10 to sample the order statistics $y^{(k_1)}, \ldots, y^{(k_r)}$ with $R = T - 1$ and use the following upper bound on $L_{P_{\mathcal{B}}\|Q_{\mathcal{B}}}$:
$$\log\left(e^{x_1/\sigma^2} + \sum_{i=1}^{r}(k_{i+1} - k_i) \cdot e^{y^{(k_i)}/\sigma^2}\right) - \log T - \frac{1}{2\sigma^2}.$$
- If $t_* \neq 1$, we can assume without loss of generality, that $t_* = 2$. In this case, we set $x_2 = y_*$. We sample $x_1 \sim \mathcal{N}(1, \sigma^2)|_{(-\infty, y_*]}$ using Fact G.3, and use a small variant of Algorithm 10 to sample the order statistics $y^{(k_1)}, \ldots, y^{(k_r)}$ with $R = T - 2$, wherein we set $y^{(k_i)} \leftarrow \Phi_\sigma^{-1}(\Phi_\sigma(y_*) \cdot \prod_{j=1}^{i} z_j)$ and use the following upper bound on $L_{P_{\mathcal{B}}\|Q_{\mathcal{B}}}(x)$:
$$\log\left(e^{x_1/\sigma^2} + e^{x_2/\sigma^2} + \sum_{i=1}^{r}(k_{i+1} - k_i) \cdot e^{y^{(k_i)}/\sigma^2}\right) - \log T - \frac{1}{2\sigma^2}.$$

G.4. **Privacy Accounting of $\mathsf{ABLQ}_{\mathcal{B}}$ for Multiple Epochs.** While the focus in this work was on the case of a *single epoch* of training (i.e., with a single pass over the training dataset), the Monte Carlo sampling approach extends to the case of k epochs since the $\delta_{\mathcal{B}}(\varepsilon) := \max\{D_{e^\varepsilon}(P_{\mathcal{B}}^{\otimes k} \parallel Q_{\mathcal{B}}^{\otimes k}), D_{e^\varepsilon}(Q_{\mathcal{B}}^{\otimes k} \parallel P_{\mathcal{B}}^{\otimes k})\}$. This can be estimated using Algorithm 12, which relies on the simple observation that

$$L_{P^{\otimes k} \parallel Q^{\otimes k}}(x^{(1)}, \ldots, x^{(k)}) = \sum_{i=1}^{k} L_{P\|Q}(x^{(i)}).$$

The order statistics sampling technique can be extended to this case by applying it independently to sample an upper bound on $L_{P\|Q}(x^{(i,j)})$ for each $j \in [k]$. Importance sampling is not directly applicable though, and we leave it to future work to construct importance samplers for the multi-epoch case.

Algorithm 12: Monte Carlo Estimation of $D_{e^\varepsilon}(P^{\otimes k} \parallel Q^{\otimes k})$.

Input: Distributions P and Q; sample access to P, Number of epochs k, Sample size m, Error probability β.
Output: An upper confidence bound on $D_{e^\varepsilon}(P^{\otimes k} \parallel Q^{\otimes k})$.
Sample $x^{(i,j)} \sim P$ for $i \in [m]$ and $j \in [k]$
$q \leftarrow \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - e^{\varepsilon - \sum_{j=1}^{k} L_{P\parallel Q}(x^{(i,j)})}\}$
$p \leftarrow$ smallest value in $[q, 1]$ such that $\mathsf{KL}(q \parallel p) \geq \log(1/\beta)/m$, or 1 if no such value exists
return p

Appendix H. Training Details

We use a neural network with five layers as the model, with around 85M parameters for the Criteo pCTR dataset and 57M parameters for the Criteo Search Conversion Logs dataset. The first layer consists of feature transforms. Categorical features are mapped into dense feature vectors using an embedding layer, with embedding dimension of 48 each. For the Criteo Search dataset, we treat all features as categorical features, whereas for the Criteo pCTR dataset, we apply a log transform for the integer features. We concatenate all the features together, and feed them into three fully connected layers with 598 hidden units each and a ReLU activation function. The last layer is a fully connected layer that gives a scalar (`logit`) prediction.

We use the Adam optimizer with a base learning rate in $\{0.0001, 0.0005, 0.001, 0.005, 0.01\}$, which is scaled with a cosine decay. We use batch sizes that are powers of 2 between 1 024 and 262 144, and we tune the norm bound $C \in \{1, 5, 10, 15, 30, 50, 100, 500, 1\,000\}$. We run the training using NVIDIA Tesla P100 GPUs, where each run takes up to an hour on a single GPU.

Since our implementation of DP-SGD in JAX works with fixed batch sizes, for each set of parameters we pick a maximum batch size B and truncate the batches to have size at most B. Batches with size smaller than B are padded with dummy examples with zero weight. For Poisson subsampling and Balls-and-Bins sampling, the batch sizes are (marginally) distributed as the binomial distribution $\mathsf{Bin}(n, b/n)$. Chua et al. [2024b, Proposition 3.2, Theorem 3.3] showed that for a given expected batch size b, the total number of examples n, the number of training steps T, and a maximum batch size of B, $\mathsf{ABLQ}_{\mathcal{P}}$ satisfies $(\varepsilon, \delta_{\mathcal{P}}(\varepsilon)+\delta')$-DP for $\delta' = (1 + e^{\varepsilon})T \cdot \Pr_{r\sim\mathrm{Bin}(n,b/n)}[r > B]$. The same argument also applies in the case of Balls-and-Bins sampling. In our experiments, we choose B such that this quantity is at most $\delta' \leq 10^{-10}$ even at $\varepsilon = 10$, so the change in the δ values is negligible relative to the values of $\delta_{\mathcal{P}}(\varepsilon)$ and $\delta_{\mathcal{B}}(\varepsilon)$ that we consider. In particular, we use maximum batch sizes in $\{1\,328, 2\,469, 4\,681, 9\,007, 17\,520, 34\,355, 67\,754, 134\,172, 266\,475\}$ for the Criteo pCTR dataset and $\{1\,320, 2\,458, 4\,665, 8\,984, 17\,488, 34\,309, 67\,687, 134\,071, 266\,317\}$ for the Criteo Search dataset, corresponding to the expected batch sizes of $\{1\,024, 2\,048, 4\,096, 8\,192, 16\,384, 32\,768, 65\,536, 131\,072, 262\,144\}$.

For the privacy accounting in Figure 6, we use order statistics sampling with the following set of order indices:

- For Criteo pCTR dataset, there are a total of 37 000 000 examples in the training set.

  - ▷ For expected batch size 1 024, there are total of $T = 36\,133$ steps. We use the order statistics of $(1, 2, \ldots, 500, 510, 520, \ldots, 1\,000, 1\,100, 1\,200, \ldots, 19\,900)$, which involves a total of 739 orders, which is about 2% of the number of steps.
  - ▷ For expected batch size 8 192, there are a total of $T = 4\,517$ steps. We use the order statistics of $(1, 2, \ldots, 500, 510, 520, \ldots, 1\,000, 1\,050, 1\,100, \ldots, 2\,950)$, which involves a total of 589 orders, which is about 13% of the number of steps.
- For Criteo Sponsored Search Conversion Log dataset, there are a total of 12 796 151 examples in the training set.
  - ▷ For expected batch size 1 024, there are total of $T = 12\,497$ steps. We use the order statistics of $(1, 2, \ldots, 500, 510, 520, \ldots, 1\,000, 1\,100, 1\,200, \ldots, 6\,900)$, which involves a total of 609 orders, which is about 4.8% of the number of steps.
  - ▷ For expected batch size 8 192, there are a total of $T = 1\,563$ steps. We use the order statistics of $(1, 2, \ldots, 500, 510, 520, \ldots, 990)$, which involves a total of 589 orders, which is about 35% of the number of steps.

For efficiency, instead of applying Monte Carlo estimation using independent samples for each ε, we instead generate $5 \cdot 10^8$ samples of upper bounds on $L_{P_{\mathcal{B}} \| Q_{\mathcal{B}}}(x)$ (resp., $L_{Q_{\mathcal{B}} \| P_{\mathcal{B}}}(x)$) using the order statistics sampling (Algorithm 10), and subsequently use them to estimate $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$ (resp., $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$). For this reason, we do not use importance sampling here since that depends on each ε. The computation was performed in parallel on a cluster of 60 CPU machines.