Balls-and-Bins Sampling for DP-SGD

Lynn Chua, Badih Ghazi, Charlie Harrison, Ethan Leeman,
Pritish Kamath, Ravi Kumar, Pasin Manurangsi, Amer Sinha, Chiyuan Zhang
Google

Abstract

We introduce the *Balls-and-Bins* sampling for differentially private (DP) optimization methods such as DP-SGD. While it has been common practice to use some form of shuffling in DP-SGD implementations, privacy accounting algorithms have typically assumed that Poisson subsampling is used instead. Recent work by Chua et al. (2024a), however, pointed out that shuffling based DP-SGD can have a much larger privacy cost in practical regimes of parameters. In this work we show that the Balls-and-Bins sampling achieves the "best-of-both" samplers, namely, the implementation of Balls-and-Bins sampling is similar to that of Shuffling and models trained using DP-SGD with Balls-and-Bins sampling achieve utility comparable to those trained using DP-SGD with Shuffling at the same noise multiplier, and yet, Balls-and-Bins sampling enjoys similar-or-better privacy amplification as compared to Poisson subsampling in practical regimes.

1 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 the training pipelines satisfy differential privacy. Since its introduction by Abadi et al. (2016), this approach of DP-SGD has been the basis of open source implementations in Tensorflow Privacy, 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., 2024).

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 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 mini-batch to have 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 scaled by the mini-batch size[1] to obtain a noisy average gradient. The privacy guarantee of the mechanism depends on the following: the noise scale σ, the number of examples in the training dataset, the size of mini-batches, the number of training steps, and the *mini-batch generation process*.

Most deep learning systems in recent years process mini-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 mini-batches. To simplify the privacy analysis, Abadi et al. (2016) considered *Poisson subsampling*, wherein each mini-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.

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

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

Adaptive Batch Linear Queries (Chua et al., 2024a). For any batch generation process, the privacy analysis of DP-SGD, particularly in the case of non-convex models such as deep neural networks, is typically done by viewing it as a post-processing of an *Adaptive Batch Linear Queries (*ABLQ*)* mechanism (Algorithm 2) that releases estimates of a sequence of adaptively chosen linear queries as produced by an *adaptive query method* $\mathcal{A}$, on the mini-batches obtained using a *batch generator* $\mathcal{G}$.

Given a dataset of n examples, the batch generator $\mathcal{G}$ can be any algorithm that generates a sequence $S_1, \ldots, S_T \subseteq [n]$ of mini-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. $\mathsf{ABLQ}_{\mathcal{G}}$ 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 average value 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 clipped gradient at the current iterate, namely $\psi_t(x) := [\nabla_{\boldsymbol{w}} \ell(\boldsymbol{w}_{t-1}, x)]_1$ (the clipping norm C is considered to be 1 w.l.o.g.), and where only the last iterate $\boldsymbol{w}_T$ is revealed.

We consider the following batch generators (see formal description in Appendix A):

- Deterministic $\mathcal{D}_{b,T}$: generates T batches each of size b in the given sequential order of the dataset,
- Shuffle $\mathcal{S}_{b,T}$: similar to $\mathcal{D}_{b,T}$, but first applies a random permutation to the dataset, and
- Poisson $\mathcal{P}_{b,T}$: each batch independently includes each example with probability $\frac{b}{n}$.

We drop the subscripts of each generator whenever it is clear from context. For any batch generator $\mathcal{G}$, e.g., $\mathcal{D}, \mathcal{P}, \mathcal{S}$, we use $\delta_{\mathcal{G}}(\varepsilon)$ to denote the *privacy loss curve* of $\mathsf{ABLQ}_{\mathcal{G}}$. Namely, for all $\varepsilon > 0$, let $\delta_{\mathcal{G}}(\varepsilon)$ be the smallest $\delta \geq 0$ such that $\mathsf{ABLQ}_{\mathcal{G}}$ satisfies (ε, δ)-DP for *all* choices of the underlying adaptive query method $\mathcal{A}$, and $\varepsilon_{\mathcal{G}}(\delta)$ is defined similarly.

It might appear that the privacy analysis of $\mathsf{ABLQ}_{\mathcal{G}}$ would be worse than that of $\mathsf{DP\text{-}SGD}_{\mathcal{G}}$ in general, since $\mathsf{ABLQ}_{\mathcal{G}}$ releases estimates to *all* intermediate linear queries, whereas, $\mathsf{DP\text{-}SGD}_{\mathcal{G}}$ only releases the *final* iterate. However, a recent interesting work of Annamalai (2024) shows that for general non-convex losses, the privacy analysis of the *last-iterate* of $\mathsf{DP\text{-}SGD}_{\mathcal{P}}$ is no better than that of $\mathsf{ABLQ}_{\mathcal{P}}$. This suggests that at least without any further assumptions on the loss functions, it might not be possible to improve the privacy analysis of $\mathsf{DP\text{-}SGD}_{\mathcal{G}}$ beyond that provided by $\mathsf{ABLQ}_{\mathcal{G}}$.

Algorithm 1 $\mathsf{DP\text{-}SGD}_{\mathcal{G}}$ (Abadi et al., 2016)

Params: Differentiable loss $\ell : \mathbb{R}^d \times \mathcal{X} \to \mathbb{R}^d$, Batch generator $\mathcal{G}_{b,T}$, initial state w_0, clipping norm C, noise scale σ.
Input: Dataset $\boldsymbol{x} = (x_1, \ldots, x_n)$.
Output: Final model state $\boldsymbol{w}_T \in \mathbb{R}^d$.

$(S_1, \ldots, S_T) \leftarrow \mathcal{G}(n)$
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
return $\boldsymbol{w}_T$

Algorithm 2 $\mathsf{ABLQ}_{\mathcal{G}}$: Adaptive Batch Linear Queries (as formalized in Chua et al. (2024a))

Params: 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 $e_t \sim \mathcal{N}(0, \sigma^2 I_d)$
return $(g_1, \ldots, g_T)$

Chua et al. (2024a) showed that the privacy guarantee of $\mathsf{ABLQ}_{\mathcal{S}}$ can be significantly worse than that of $\mathsf{ABLQ}_{\mathcal{P}}$ in practical regimes of privacy parameters, especially for small noise scale σ; subsequently such gaps were also observed in empirical privacy by Annamalai et al. (2024). This seriously challenges the common practice of using shuffling in DP-SGD while claiming privacy guarantees based on Poisson subsampling. Concurrently, Lebeda et al. (2024) consider the batch generator $\mathcal{W}$ where each batch is generated independently by sampling examples from the dataset without replacement, until the desired batch size is met, and show that $\mathsf{ABLQ}_{\mathcal{W}}$ also exhibits worse privacy guarantees compared to $\mathsf{ABLQ}_{\mathcal{P}}$. The central question we consider is:

Is there a batch generator that is similar to Shuffling in terms of implementation simplicity and model utility, but has favorable privacy analysis, namely similar to or better than Poisson subsampling?

Our Contributions. Towards answering this question, we introduce the *Balls-and-Bins* generator $\mathcal{B}_T$ (Algorithm 3), which operates by placing each example in a random batch. This sampler exhibits behavior similar to Shuffle, with each example appearing in precisely one batch, and is also similar to imple-

Algorithm 3 $\mathcal{B}_T$: Balls-and-Bins Sampler

Params: Number of batches T.
Input: Number of datapoints n.
Output: Seq. 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**
    $S_t \leftarrow S_t \cup \{i\}$ for uniformly random $t \in [T]$
  return $S_1, \ldots, S_T$

ment. On the other hand, the marginal distribution over each batch is exactly the same as that of Poisson subsampling. This overcomes the privacy lower bound of Chua et al. (2024a); Lebeda et al. (2024) by making the positions of the different examples to be independent, thereby preventing non-differing examples from leaking information about the presence or absence of the differing example in any given batch.

We identify a tightly dominating pair (Definition 2.2) for $\mathsf{ABLQ}_{\mathcal{B}}$, thereby allowing a tight privacy analysis. We show that $\mathsf{ABLQ}_{\mathcal{B}}$ enjoys better privacy guarantees compared to $\mathsf{ABLQ}_{\mathcal{D}}$ and $\mathsf{ABLQ}_{\mathcal{S}}$ in all regime of parameters. This is in sharp contrast to $\mathsf{ABLQ}_{\mathcal{P}}$, which can have worse privacy guarantees than even $\mathsf{ABLQ}_{\mathcal{D}}$ at large ε (Chua et al., 2024a).

We use a Monte Carlo method for estimating $\delta_{\mathcal{B}}(\varepsilon)$. However, naive Monte Carlo methods are inefficient when estimating small values of δ, or when the number of steps T is large. Our key contributions here are to develop the techniques of *importance sampling*, to handle small δ values, and *order statistics sampling*, a new technique to handle a large number of steps. We believe the latter is of independent interest beyond DP.

Finally, 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.

Related Work. Balle et al. (2020) considered a model of *random check-ins* in the context of distributed (a.k.a. federated) learning, which is essentially the same as balls-and-bins sampling. Their analysis however relies on the amplification properties of shuffling and does not lead to better privacy guarantees than those known for shuffling.

An independent and concurrent work of Choquette-Choo et al. (2025) also considered the Balls-and-Bins sampling method, although in the context of the so-called DP-FTRL algorithm, the variant of DP-SGD that adds correlated noise at each step (Kairouz et al., 2021; McMahan et al., 2022). They also use a Monte Carlo method to compute the privacy parameters. Since DP-SGD is a special case of DP-FTRL (with independent noise), the dominating pair we identify is a special case of the dominating pair for DP-FTRL, which depends on the specific correlation matrix. Choquette-Choo et al. (2025) mention "a more careful sampler and concentration analysis" as an avenue for improving the sample complexity of the Monte Carlo estimation. In our work, we develop techniques based on importance sampling and order statistics sampling to improve the cost of the Monte Carlo method. Extending these techniques to the general DP-FTRL setting is an interesting future direction.

2 PRELIMINARIES

A *mechanism* $\mathcal{M}$ maps input datasets to distributions over an output space. Namely, for $\mathcal{M} : \mathcal{X}^* \to \Delta_{\mathcal{O}}$, on input *dataset* $\boldsymbol{x} = (x_1, \ldots, x_n)$ where each *record* $x_i \in \mathcal{X}$, $\mathcal{M}(\boldsymbol{x}) \in \Delta_{\mathcal{O}}$ 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. 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 2.1 (DP)**.** 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 $\Pr[\mathcal{M}(\boldsymbol{x}) \in E] \leq e^{\varepsilon} \Pr[\mathcal{M}(\boldsymbol{x}') \in E] + \delta$.

Following Kairouz et al. (2021); Chua et al. (2024a), we use the "zeroing-out" adjacency. Consider the augmented input space $\mathcal{X}_{\perp} := \mathcal{X} \cup \{\perp\}$ 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 $\perp$. We use $\boldsymbol{x} \to_z \boldsymbol{x}'$ to specifically emphasize that $x_i \in \mathcal{X}$ and $x'_i = \perp$. Thus, $\boldsymbol{x} \sim \boldsymbol{x}'$ if $\boldsymbol{x} \to_z \boldsymbol{x}'$ or $\boldsymbol{x}' \to_z \boldsymbol{x}$.

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 $D_{e^{\varepsilon}}(\mathcal{M}(\boldsymbol{x}) \parallel \mathcal{M}(\boldsymbol{x}')) \leq \delta$.

Definition 2.2 (Dominating Pair (Zhu et al., 2022)). The pair (P, Q) *dominates* the pair (A, B) (denoted $(P, Q) \succcurlyeq (A, B)$) if $D_{e^\varepsilon}(P \parallel Q) \geq D_{e^\varepsilon}(A \parallel B)$ 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 $\varepsilon \geq 0$, it holds that $\delta_{\mathcal{M}}(\varepsilon) \leq \max\{D_{e^\varepsilon}(P \parallel Q), D_{e^\varepsilon}(Q \parallel P)\}$, and conversely, if $\mathcal{M} \succcurlyeq (P, Q)$, then for all $\varepsilon \geq 0$, it holds that $\delta_{\mathcal{M}}(\varepsilon) \geq \max\{D_{e^\varepsilon}(P \parallel Q), D_{e^\varepsilon}(Q \parallel P)\}$.[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).[4]

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 implies the following.

Lemma 2.3. *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 $(P, Q) \succcurlyeq (A, B)$.*[5]

Dominating Pairs for $\mathsf{ABLQ}_{\mathcal{G}}$. Recall again, that we use $\mathcal{D}$, $\mathcal{P}$, $\mathcal{S}$ to refer to the deterministic, Poisson, and shuffle batch generators respectively. Tightly dominating pairs are known for $\mathsf{ABLQ}_{\mathcal{D}}$ and $\mathsf{ABLQ}_{\mathcal{P}}$:

Proposition 2.4 (Balle and Wang (2018, Theorem 8)). *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 $Q_{\mathcal{D}} := \mathcal{N}(0, \sigma^2)$.*

Proposition 2.5 (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*

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

These tightly dominating pairs enable numerical computation of the privacy parameters. Namely, $\delta_{\mathcal{D}}(\varepsilon)$ can be computed easily using Gaussian CDFs, whereas $\delta_{\mathcal{P}}(\varepsilon)$ can be computed using numerical methods with the Fast-Fourier transform, as provided in multiple open source libraries (Prediger and Koskela, 2020; Microsoft., 2021; Google's DP Library., 2020).

On the other hand, while it is unclear whether a tightly dominating pair for $\mathsf{ABLQ}_{\mathcal{S}}$ even exists, Chua et al. (2024a) studied a pair that is dominated by $\mathsf{ABLQ}_{\mathcal{S}}$, while conjecturing it to be tightly dominating.

Proposition 2.6 (Chua et al. (2024a)). *For all $\sigma > 0$ and $T \geq 1$, it holds that $\mathsf{ABLQ}_{\mathcal{S}} \succcurlyeq (P_{\mathcal{S}}, Q_{\mathcal{S}})$ for*

$$P_{\mathcal{S}} := \sum_{i=1}^{T} \tfrac{1}{T} \cdot \mathcal{N}(2e_i, \sigma^2 I_T),$$
$$Q_{\mathcal{S}} := \sum_{i=1}^{T} \tfrac{1}{T} \cdot \mathcal{N}(e_i, \sigma^2 I_T).$$

where $e_i \in \mathbb{R}^T$ is the ith standard basis vector, and I_T is the $T \times T$ identity matrix.

3 BALLS-AND-BINS SAMPLER

We identify a tightly dominating pair for $\mathsf{ABLQ}_{\mathcal{B}}$.

Theorem 3.1. *For all $\sigma > 0$ and $T \geq 1$, it holds that $(P_{\mathcal{B}}, Q_{\mathcal{B}}) \equiv \mathsf{ABLQ}_{\mathcal{B}}$ for*[6]

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

The proof relies on the well-known "joint convexity" property of the hockey stick divergence.

Proposition 3.2 (Joint Convexity of Hockey Stick Divergence; see, e.g., Lemma B.1 in Chua et al. (2024a)). *Given two 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 3.1. To show $\mathsf{ABLQ}_{\mathcal{B}} \succcurlyeq (P_{\mathcal{B}}, Q_{\mathcal{B}})$, 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 adjacent datasets $\boldsymbol{x} \to_z \boldsymbol{x}'$ that differ on say example x_n

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

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

[4] Our terminology is slightly different from Zhu et al. (2022) in that they refer to (P, Q) as a *tightly dominating pair* if $\delta_{\mathcal{M}}(\varepsilon) = D_{e^\varepsilon}(P \| Q)$ 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. This is true, e.g. for even one step of $\mathsf{ABLQ}_{\mathcal{P}}$. Our notation is asymmetric in defining a dominating pair, allowing a tight characterization of $\mathsf{ABLQ}_{\mathcal{P}}$.

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

[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 discuss the difficulty of numerically approximating its hockey stick divergence.

and $x'_n = \perp$. For $\Gamma = (S'_1, \ldots, S'_T)$ for $S'_i \subseteq [n-1]$ being an assignment of batches for all other examples, 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 Lemma 2.3 and Proposition 3.2 that $(P_\mathcal{B}, Q_\mathcal{B}) \not\succcurlyeq \mathsf{ABLQ}_\mathcal{B}$.

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 $e_t \sim \mathcal{N}(0, \sigma^2(I - \psi_t(x_n)\psi_t(x_n)^\top))$; this is inductively defined since ψ_t potentially depends on $g_1, \ldots, g_{t-1}$.

First let us consider the case when $v \sim Q_\mathcal{B} = \mathcal{N}(0, \sigma^2 I)$. In this case, $\psi_t(x_n)v_t + e_t$ is distributed precisely as $\mathcal{N}(0, \sigma^2 I)$, and thus, g_t is distributed as $\sum_{i \in S'_t} \psi_t(x_i) + e'_t$ for $e'_t \sim \mathcal{N}(0, \sigma^2 I)$ 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 $v \sim \mathcal{N}(e_{t_*}, \sigma^2 I)$. For a fixed t_* and sampling $v \sim \mathcal{N}(e_{t_*}, \sigma^2 I)$, g_t is distributed as $\sum_{i \in S'_t} \psi_t(x_i) + e'_t$ for $e'_t \sim \mathcal{N}(0, \sigma^2 I)$ for $t \neq t_*$ and distributed as $\sum_{i \in S'_t} \psi_t(x_i) + \psi_t(x_n) + e'_t$ for $e'_t \sim \mathcal{N}(0, \sigma^2 I)$ 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 $f(P_\mathcal{B}) = \mathsf{ABLQ}_\mathcal{B}(\boldsymbol{x})$. □

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

$\delta_\mathcal{B}$ vs. $\delta_\mathcal{D}$ and $\delta_\mathcal{S}$. A simple consequence of Proposition 3.2 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 3.3. *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 3.1 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 $\mathsf{ABLQ}_\mathcal{S} \not\succcurlyeq (P_\mathcal{B}, Q_\mathcal{B})$ and hence $\delta_\mathcal{B}(\varepsilon) \leq \delta_\mathcal{S}(\varepsilon)$. □

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

Theorem 3.4. *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. Chua et al. (2024a, Theorem 4.2) showed that 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 3.3 gives us that $\delta_\mathcal{B}(\varepsilon) < \delta_\mathcal{P}(\varepsilon)$ for all $\varepsilon \geq \varepsilon_0$. □

Remark 3.5. *In Appendix D, 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 3.4. Thus, the privacy guarantees of $\mathsf{ABLQ}_\mathcal{P}$ and $\mathsf{ABLQ}_\mathcal{B}$ are in general incomparable.*

4 ESTIMATING $\delta_\mathcal{B}(\varepsilon)$

Hockey stick divergence $D_\varepsilon(P \parallel Q)$ can be expressed in terms of the *privacy loss function* $L_{P \parallel Q}(x)$ (Dwork and Rothblum, 2016)[7] as

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

where $L_{P \parallel Q}(x) := \log \frac{P(x)}{Q(x)}$,[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 $L_{P_\mathcal{B} \parallel Q_\mathcal{B}}(x)$ 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:

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

$$L_{Q_\mathcal{B} \parallel P_\mathcal{B}}(x) = \log T + \tfrac{1}{2\sigma^2} - \log\left(\sum_{t=1}^T e^{x_t/\sigma^2}\right). \tag{3}$$

Monte Carlo Estimation. (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 $L_{P \parallel Q}(x)$; such an approach was previously studied by Wang et al. (2023). We can obtain high probability upper bounds via the Chernoff–Hoeffding bound as described in Algorithm 4, 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 $\mathsf{Ber}(q)$.

[7] The distribution of $L_{P \parallel Q}(x)$ 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 4 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 $x^{(1)}, \dots, x^{(m)} \sim P$
$q \leftarrow \frac{1}{m}\sum_{i=1}^m \max\{0, 1 - e^{\varepsilon - L_{P \parallel Q}(x^{(i)})}\}$
$p \leftarrow$ 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

Theorem 4.1. *For all distributions P and Q, Algorithm 4 returns an upper bound on $D_{e^\varepsilon}(P \parallel Q)$ with probability $1 - \beta$.*

Fact 4.2 (Hoeffding (1963))**.** *For $Z_1, \dots, Z_m$ drawn i.i.d. from distribution P over $[0, 1]$ with mean μ,*

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

Proof of Theorem 4.1. Suppose $D_{e^\varepsilon}(P, Q) = \mu$. If the returned value p by Algorithm 4 is 1, then $\mu \leq 1$ holds trivially. Otherwise if p is smaller than μ, then we have that the empirical average q is such that $\mathsf{KL}(q \parallel \mu) > \mathsf{KL}(q \parallel p) \geq \log(1/\beta)/m$. From Fact 4.2, this can happen with probability at most β. □

Algorithm 4 in principle allows us to obtain an upper bound on the hockey stick divergence to arbitrary accuracy and with arbitrarily high probability through the guarantees in Theorem 4.1 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 method of "order statistics sampling" respectively; we believe the latter could be of independent interest.

Before describing these methods, we first note a simplification when estimating $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$ using Algorithm 4. For $P_t := \mathcal{N}(e_t, \sigma^2 I)$, we have that $P_{\mathcal{B}} = \sum_{t=1}^T \frac{1}{T} P_t$. 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 4.

Importance Sampling. We provide a generic approach for improving the sample complexity of Algorithm 4 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 (1), as explained in Algorithm 5. 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 $C_\varepsilon = \frac{1}{2} - \varepsilon\sigma^2$. The choice of C_ε is such that for all $x \notin E_\varepsilon$,

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

We show how to efficiently sample from $Q_{\mathcal{B}}|_{x\in E_\varepsilon}$ in Appendix B.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 $x_1 \leq C_\varepsilon + 1$ and $x_t \leq C_\varepsilon$ for all $t > 1$, and hence

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

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 B.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 B.1.

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 $k_1, \dots, k_r \in \{1, \dots, R\}$ of increasing indices, the following holds, where we use $k_{r+1} := R$ and $k_0 := 0$ for convenience. Given $x_1, \dots, x_R \in \mathbb{R}$, let $y^{(1)}, \dots, y^{(R)}$ denote the same values as x_i's but in sorted order $y^{(1)} \geq \cdots \geq y^{(R)}$. Then,

$$\textstyle\sum_{t=1}^R e^{x_t/\sigma^2} \leq \sum_{i=1}^r (k_{i+1} - k_i) \cdot e^{y^{(k_i)}/\sigma^2} \quad (4)$$

$$\textstyle\sum_{t=1}^R e^{x_t/\sigma^2} \geq \sum_{i=1}^r (k_i - k_{i-1}) \cdot e^{y^{(k_i)}/\sigma^2}, \quad (5)$$

Algorithm 5 Monte Carlo Estimation of $D_{e^\varepsilon}(P \parallel Q)$ with Importance Sampling.

Input: Distributions P and Q, Event E such that $L_{P \parallel Q}(x) < \varepsilon$ 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 $x^{(1)}, \ldots, x^{(m)} \sim P|_{x \in E}$
$q \leftarrow \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - e^{\varepsilon - L_{P \parallel Q}(x^{(i)})}\}$
$p \leftarrow$ 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 $p \cdot P(E)$

where (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 (4) and $k_1 = 1$ and $k_2 = R$ for (5).

Our key idea for efficient sampling 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 6 efficiently samples order statistics of P, wherein $\mathsf{Beta}(a, b)$ is the Beta distribution over $[0, 1]$. Theorem B.5 (Appendix B.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 $L_{Q_{\mathcal{B}} \parallel P_{\mathcal{B}}}(x)$ as

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

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

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

where we sample $x_1 \sim \mathcal{N}(1, \sigma^2)$ and $y^{(k_1)}, \ldots, y^{(k_r)}$ using Algorithm 6 for $R = T - 1$, and plug them into Algorithm 4. While not immediate, this can also be used in conjunction with importance sampling as in Algorithm 5. We defer the details to Appendix B.3.

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, inspired by Chua et al. (2024a). The idea is to use the following lower bound

$$\delta_{\mathcal{B}}(\varepsilon) \geq D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}}) \geq \sup_C P_{\mathcal{B}}(S_C) - e^\varepsilon Q_{\mathcal{B}}(S_C), \tag{6}$$

where $S_C := \{x : \max_t x_t \geq C\}$; the main reason being that $P_{\mathcal{B}}(S_C)$ and $Q_{\mathcal{B}}(S_C)$ are efficiently computable.

Algorithm 6 Sampling Order Statistics of P

Input: Number R of random variables $\sim P$, Orders $k_1, \ldots, k_r \in \{1, \ldots, R\}$
Output: $(y^{(k_1)}, \ldots, y^{(k_r)})$ jointly distributed as $(k_1, \ldots, k_r)$ order statistics of R draws from P

Let $k_0 \leftarrow 0$
for $i = 1, \ldots, r$ **do**
  $z_i \sim \mathsf{Beta}(R - k_i + 1, k_i - k_{i-1})$
  $y^{(k_i)} \leftarrow \mathsf{CDF}_P^{-1}(\prod_{j=1}^{i} z_j)$
return $(y^{(k_1)}, \ldots, y^{(k_r)})$

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 4; see, e.g., Figure 2.

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.

5 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 samplers from the privacy accounting.

Implementation Details. We use the scalable batch sampling approach proposed by Chua et al. (2024b) using massively parallel computation (Dean and Ghemawat, 2004) for sampling batches using each of the batch generators. We use JAX (Bradbury et al., 2018) for training neural networks. Since it is more efficient to have fixed batch sizes, we follow Chua et al. (2024b) and fix a certain maximum batch size B when using Balls-and-Bins and Poisson subsampling, and for batches that exceed size B, we randomly subsample without replacement to get a batch of size B. This can be done by incurring a small penalty in the privacy parameters, as described in Appendix C. For batches that are smaller than size B, we pad with examples with a weight of 0 so that the batch size is exactly B. We use a weighted loss function so that the mini-batch loss is unaffected by the padding. We note that other optimizations could also be possible, such as using accumulation of gradients computed in small physical batches (Abadi et al., 2016; Beltran et al., 2024).

Finally, we note Balls-and-Bins can be implemented trivially given any implementation of Shuffling. Namely, given examples $x_1, \ldots, x_n$ in a ran-

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

Datasets. The first dataset we use 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 a click on an ad 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 C.

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 1. 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 samplers 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 2. 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 a lower bound as shown by Chua et al. (2024a). For $\delta_{\mathcal{B}}$, we plot a lower bound from (6), the mean of the Monte Carlo estimate (value q in Algorithm 4) and the upper confidence bound (value p in Algorithm 4) 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.

6 DISCUSSION

We introduce the Balls-and-Bins sampler for DP-SGD, and showed 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 B.4.

Our work leaves several directions open for future investigation. The main open problem is to obtain a tight provable privacy accounting for $\mathsf{ABLQ}_{\mathcal{B}}$, unlike the high probability bounds that we establish or to establish through some approximation that it is no worse than $\mathsf{ABLQ}_{\mathcal{P}}$ in relevant 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 Rényi DP guarantees (Mironov, 2017).

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

Acknowledgments

We thank anonymous reviewers 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.

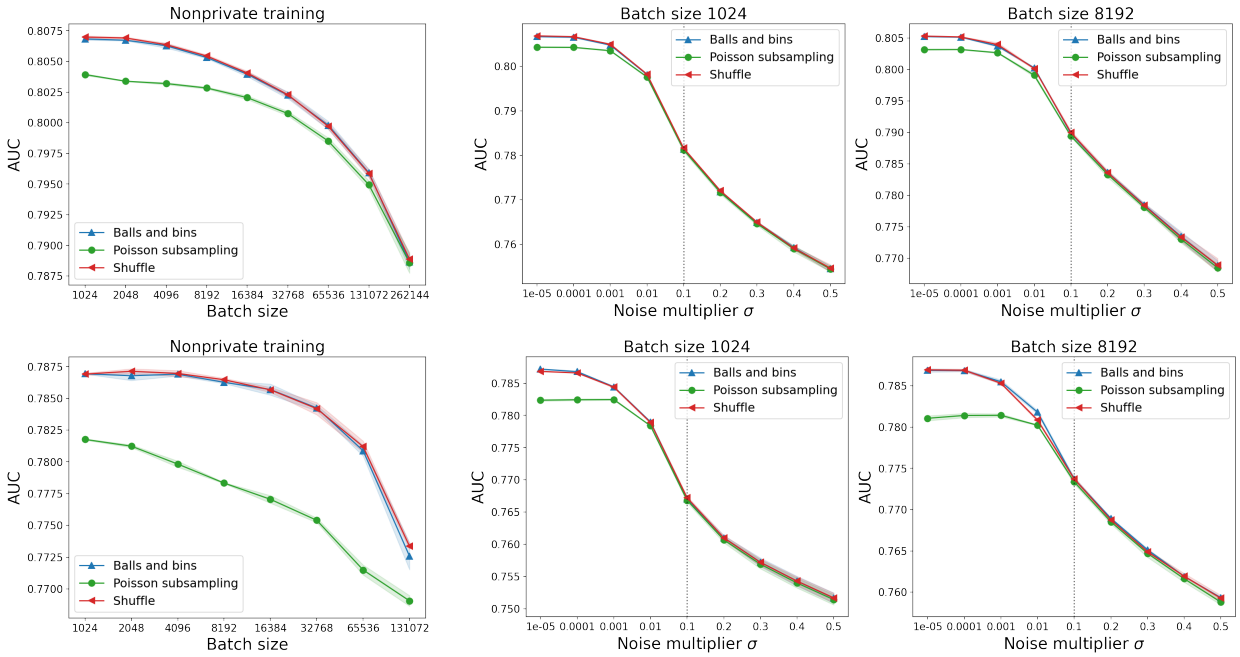

Figure 1: AUC values for 1 epoch of training with the Criteo Display Ads pCTR dataset (top) and the Criteo Sponsored Search Conversion Log dataset (bottom). On the left, we train without privacy and vary the batch size. In the middle and right, we train privately with varying σ, using (expected) batch sizes 1024 (middle) and 8192 (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.

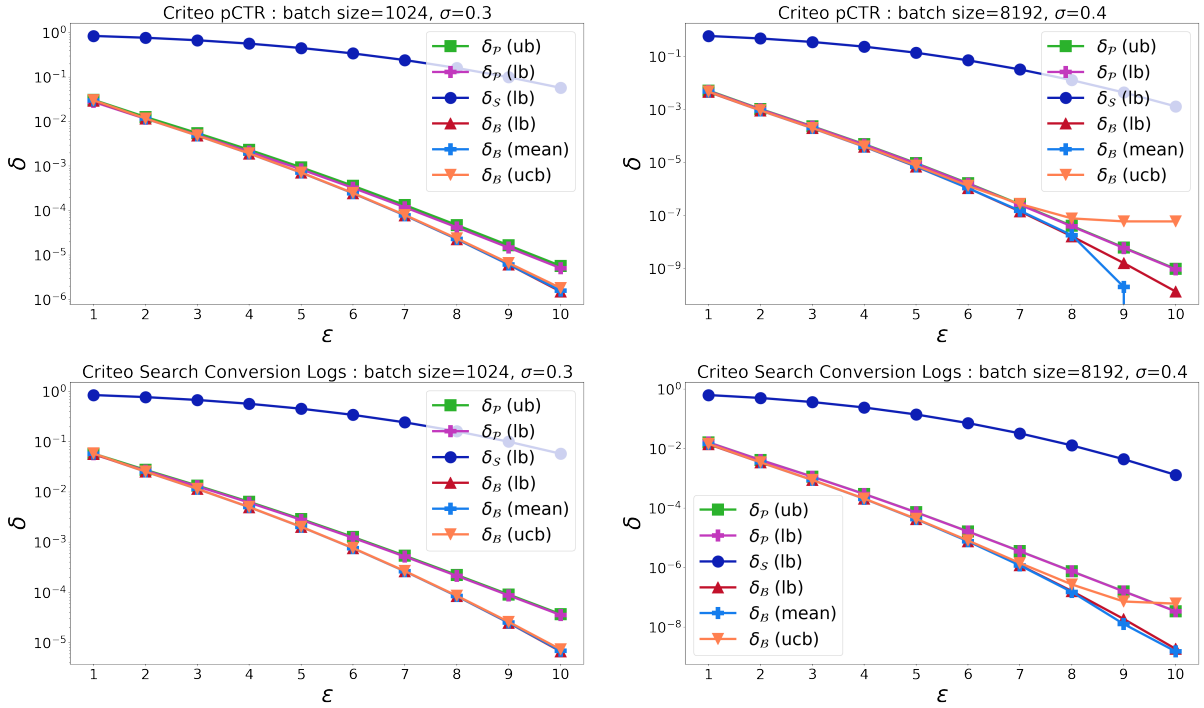

Figure 2: Bounds on $\delta_{\mathcal{P}}$, $\delta_{\mathcal{S}}$, and $\delta_{\mathcal{B}}$ are plotted for various values of ε for different (expected) batch size and σ. These mean and upper confidence bounds for $\delta_{\mathcal{B}}$ were obtained using order statistics sampling (specific orders and sample complexity specified in Appendix C).

References

Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *CCS*, pages 308–318, 2016.

Rohan Anil, Badih Ghazi, Vineet Gupta, Ravi Kumar, and Pasin Manurangsi. Large-scale differentially private BERT. In *EMNLP (Findings)*, 2022.

Meenatchi Sundaram Muthu Selva Annamalai. It's our loss: No privacy amplification for hidden state DP-SGD with non-convex loss. In *AISec*, pages 24–30, 2024.

Meenatchi Sundaram Muthu Selva Annamalai, Borja Balle, Emiliano De Cristofaro, and Jamie Hayes. To shuffle or not to shuffle: Auditing DP-SGD with shuffling. *CoRR*, abs/2411.10614, 2024.

Borja Balle and Yu-Xiang Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *ICML*, 2018.

Borja Balle, Peter Kairouz, Brendan McMahan, Om Thakkar, and Abhradeep Guha Thakurta. Privacy amplification via random check-ins. In *NeurIPS*, pages 4623–4634, 2020.

Borja Balle, Leonard Berrada, Soham De, Sahra Ghalebikesabi, Jamie Hayes, Aneesh Pappu, Samuel L Smith, and Robert Stanforth. JAX-Privacy: Algorithms for privacy-preserving machine learning in JAX, 2022. URL http://github.com/google-deepmind/jax_privacy.

Sebastian Rodriguez Beltran, Marlon Tobaben, Joonas Jälkö, Niki Loppi, and Antti Honkela. Towards efficient and scalable training of differentially private deep learning. In *NeurIPS*, 2024.

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

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Christopher A. Choquette-Choo, Arun Ganesh, Saminul Haque, Thomas Steinke, and Abhradeep Thakurta. Near exact privacy amplification for matrix mechanisms. In *ICLR*, 2025.

Lynn Chua, Badih Ghazi, Pritish Kamath, Ravi Kumar, Pasin Manurangsi, Amer Sinha, and Chiyuan Zhang. How private are DP-SGD implementations? In *ICML*, 2024a.

Lynn Chua, Badih Ghazi, Pritish Kamath, Ravi Kumar, Pasin Manurangsi, Amer Sinha, and Chiyuan Zhang. Scalable DP-SGD: Shuffling vs. Poisson subsampling. In *NeurIPS*, 2024b.

Herbert A David and Haikady N Nagaraja. *Order Statistics*. John Wiley & Sons, 2004.

Soham De, Leonard Berrada, Jamie Hayes, Samuel L. Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale. *CoRR*, abs/2204.13650, 2022.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

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

Tim Dockhorn, Tianshi Cao, Arash Vahdat, and Karsten Kreis. Differentially private diffusion models. *TMLR*, 2023.

Cynthia Dwork and Guy N. Rothblum. Concentrated differential privacy. *CoRR*, abs/1603.01887, 2016.

Vitaly Feldman and Moshe Shenfeld. Privacy amplification by random allocation. *CoRR*, abs/2502.08202, 2025.

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

Jiyan He, Xuechen Li, Da Yu, Huishuai Zhang, Janardhan Kulkarni, Yin Tat Lee, Arturs Backurs, Nenghai Yu, and Jiang Bian. Exploring the limits of differentially private deep learning with group-wise clipping. In *ICLR*, 2023.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *J. ASA*, 58(301):13–30, 1963.

Timour Igamberdiev, Doan Nam Long Vu, Felix Künnecke, Zhuo Yu, Jannik Holmer, and Ivan Habernal. DP-NMT: Scalable differentially-private machine translation. In *EACL (Demonstrations)*, pages 94–105, 2024.

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

Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. In *ICML*, pages 1376–1385, 2015.

Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu.

Practical and private (deep) learning without sampling or shuffling. In *ICML*, pages 5213–5225, 2021.

Antti Koskela, Joonas Jälkö, and Antti Honkela. Computing tight differential privacy guarantees using FFT. In *AISTATS*, pages 2560–2569, 2020.

Antti Koskela, Mikko A. Heikkilä, and Antti Honkela. Numerical accounting in the shuffle model of differential privacy. *TMLR*, 2023, 2023.

Christian Janos Lebeda, Matthew Regehr, and Gautam Kamath. Avoiding pitfalls for privacy accounting of subsampled mechanisms under composition. *CoRR*, abs/2405.20769, 2024.

George Marsaglia and Wai Wan Tsang. A simple method for generating gamma variables. *ACM Trans. Math. Softw.*, 26(3):363–372, 2000.

Brendan McMahan, Keith Rush, and Abhradeep Guha Thakurta. Private online prefix sums via optimal matrix factorizations. *CoRR*, abs/2202.08312, 2022.

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.

Ilya Mironov. Rényi differential privacy. In *CSF*, pages 263–275, 2017.

Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H. Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Guha Thakurta. How to dp-fy ML: A practical guide to machine learning with differential privacy. *J. AIR*, 77:1113–1201, 2023.

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

Marcelo Tallis and Pranjul Yadav. Reacting to variations in product demand: An application for conversion rate (CR) prediction in sponsored search. *CoRR*, abs/1806.08211, 2018.

Xinyu Tang, Ashwinee Panda, Milad Nasr, Saeed Mahloujifar, and Prateek Mittal. Private fine-tuning of large language models with zeroth-order optimization. *CoRR*, abs/2401.04343, 2024.

Tensorflow Privacy, 2024. URL https://www.tensorflow.org/responsible_ai/privacy/api_docs/python/tf_privacy.

Jiachen (Tianhao) Wang, Saeed Mahloujifar, Tong Wu, Ruoxi Jia, and Prateek Mittal. A randomized approach to tight privacy accounting. In *NeurIPS*, pages 33856–33893, 2023.

Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-friendly differential privacy library in PyTorch. *CoRR*, abs/2109.12298, 2021.

Yuqing Zhu, Jinshuo Dong, and Yu-Xiang Wang. Optimal accounting of differential privacy via characteristic function. In *AISTATS*, pages 4782–4817, 2022.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. **Yes**
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. **Yes**
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. **Yes**: An implementation of the privacy accounting algorithms introduced in this work are available at github.com/google-research/google-research/tree/master/dpsgd_batch_sampler_accounting.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. **Yes**
   (b) Complete proofs of all theoretical results. **Yes**
   (c) Clear explanations of any assumptions. **Yes**

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). **Yes**: An implementation of the privacy accounting algorithms introduced in this work are available at github.com/google-research/google-research/tree/master/dpsgd_batch_sampler_accounting. We do not include the code needed to evaluate our experiments with DP-SGD on neural network architectures, as they are orthogonal to the main contributions of this work.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). **Yes**: Experimental details are presented in Appendix C.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). **Yes**
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). **Yes**: We include details in Appendix C.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. **Not Applicable**
   (b) The license information of the assets, if applicable. **Not Applicable**
   (c) New assets either in the supplemental material or as a URL, if applicable. **Not Applicable**
   (d) Information about consent from data providers/curators. **Not Applicable**
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. **Not Applicable**

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. **Not Applicable**
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. **Not Applicable**
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. **Not Applicable**

Balls-and-Bins Sampling for DP-SGD: Supplementary Material

A Batch Generators

We formally describe the Deterministic ($\mathcal{D}$), Shuffle ($\mathcal{S}$), and Poisson ($\mathcal{P}$) batch generators considered in this work, as formalized in Chua et al. (2024a). Let n be the number of datapoints.

- Deterministic $\mathcal{D}_{b,T}$, formalized in Algorithm 7, generates T batches each of size b in the given sequential order of the dataset. This method requires that $n = b \cdot T$.
- Shuffle $\mathcal{S}_{b,T}$, formalized in Algorithm 8, is similar to $\mathcal{D}_{b,T}$, but first applies a random permutation to the dataset. This method also requires $n = b \cdot T$.
- Poisson $\mathcal{P}_{b,T}$, formalized in Algorithm 9, samples each batch independently by including each example with probability $\frac{b}{n}$. This method works for any n and results in an expected batch size of b.

Algorithm 7 $\mathcal{D}_{b,T}$: Deterministic Batch Generator

Params: Batch size b, number of batches T.
Input: Number of datapoints $n = b \cdot T$.
Output: Seq. of disjoint batches $S_1, \ldots, S_T \subseteq [n]$.

for $t = 0, \ldots, T-1$ **do**
  $S_{t+1} \leftarrow \{tb+1, \ldots, tb+b\}$
return $S_1, \ldots, S_T$

Algorithm 8 $\mathcal{S}_{b,T}$: Shuffle Batch Generator

Params: Batch size b, number of batches T.
Input: Number of datapoints $n = b \cdot T$.
Output: Seq. of disjoint batches $S_1, \ldots, S_T \subseteq [n]$.

$\pi \leftarrow$ random permutation over $[n]$
for $t = 0, \ldots, T-1$ **do**
  $S_{t+1} \leftarrow \{\pi(tb+1), \ldots, \pi(tb+b)\}$
return $S_1, \ldots, S_T$

Algorithm 9 $\mathcal{P}_{b,T}$: Poisson Batch Generator

Params: Expected batch size b, num. of batches T.
Input: Number of datapoints n.
Output: Seq. 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**

$$S_t \leftarrow \begin{cases} S_t \cup \{i\} & \text{with probability } b/n \\ S_t & \text{with probability } 1 - b/n \end{cases}$$

return $S_1, \ldots, S_T$

We recall that since we are using the "zeroing-out" adjacency, n is known, and not protected under DP.

B Importance and Order Statistics Sampling

We describe how to efficiently perform importance sampling as described in Algorithm 5 for the pair $(P_{\mathcal{B}}, Q_{\mathcal{B}})$ as well as the proof that Algorithm 6 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 $\mathsf{CDF}_P(x) := \Pr_{z\sim P}[z \le x]$ 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 B.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 B.2 (Order Statistics and Beta Distribution)**.** *The random variable $y^{(k)}$ that is the kth largest element among $x_1,\dots,x_R \sim \mathsf{Unif}[0,1]$ 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 B.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 B.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.

Fact B.2 and B.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 B.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 $y^{(k)}$.

B.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 5. We use $\Phi_\sigma(\cdot)$ to denote $\mathsf{CDF}_{\mathcal{N}(0,\sigma^2)}$ for short.

Estimating $D_{e^\varepsilon}(Q_{\mathcal{B}} \parallel P_{\mathcal{B}})$. Recall that 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 $C_\varepsilon := \frac{1}{2} - \varepsilon\sigma^2$. 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 B.3, by sampling $y_t \sim \mathsf{Unif}[0,\Phi_\sigma(C_\varepsilon)]$ and returning $x_t = \Phi_\sigma^{-1}(y_t)$ for each $t \in [T]$.

[9] In cases where $\mathsf{CDF}_P(\cdot)$ is not a continuous function, the inverse is defined as $\mathsf{CDF}_P^{-1}(y) := \min_{x\in\mathbb{R}:\mathsf{CDF}_P(x)\ge y} x$; 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.

Algorithm 10 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 B.3)
  $t_* \sim$ uniformly random coordinate in $[T]$
  for $t \in \{1,\dots,T\}$ **do**
    if $t = t_*$ **then**
      $z_t \leftarrow y_*$
    else
      $z_t \sim \mathsf{Unif}[0, y_*]$
  return $(\mathsf{CDF}_P^{-1}(z_t) : t \in [T])$

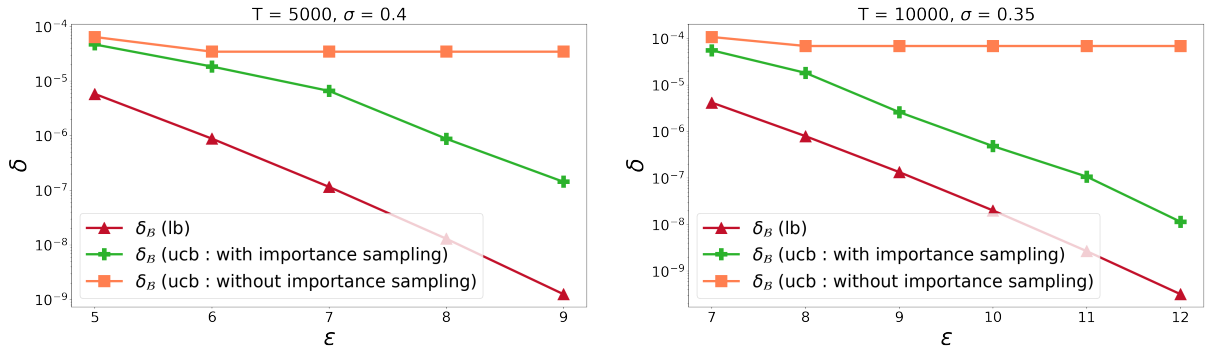

Figure 3: Upper confidence bounds on $\delta_{\mathcal{B}}(\varepsilon)$ against various values of ε for two settings of T and σ, with and without importance sampling. Additionally, lower bounds on $\delta_{\mathcal{B}}(\varepsilon)$ are included.

Estimating $D_{e^\varepsilon}(P_{\mathcal{B}} \parallel Q_{\mathcal{B}})$. Recall that 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 $P_1 = \mathcal{N}(e_1, \sigma^2 I)$ and $E_\varepsilon := \{x : \max\{x_1 - 1, \max_{t>1} x_t\} \geq C_\varepsilon\}$ for $C_\varepsilon = \frac{1}{2} + \sigma^2 \cdot \left(\varepsilon - \log\left(1 + \frac{e^{1/\sigma^2}-1}{T}\right)\right)$. 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 $\mathcal{N}(0, \sigma^2 I_T)|_{\{x : \max_t x_t \geq C_\varepsilon\}}$. In Algorithm 10, we provide a generic algorithm that for any distribution P over $\mathbb{R}$, samples from the distribution $P^{\otimes T}|_{\{x : \max_t x_t \geq C\}}$, 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 $x' \sim \mathcal{N}(0, \sigma^2 I_T)|_{\{x' : \max_t x'_t \geq C_\varepsilon\}}$ using Algorithm 10, and returning $x = x' + e_1$.

Numerical Evaluation. To demonstrate the usefulness of our importance sampling method, in Figure 3, we plot the upper confidence bound on $\delta_{\mathcal{B}}(\varepsilon)$ as obtained via Algorithm 4 (i.e., without importance sampling) and via Algorithm 5 (i.e., with importance sampling) along the lower bound obtained via (6). The upper confidence bounds are obtained for error probability $\beta = 10^{-3}$. For a similar running time, we see that Algorithm 5 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 3 for $T = 5000$ and $\sigma = 0.4$, at $\varepsilon = 12$, 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 $1/P_{\mathcal{B}}(E_\varepsilon)$.

B.2 Order Statistics Sampling

We show that Algorithm 6 indeed samples from the joint distribution of order statistics of P.

Theorem B.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 6 are distributed as the $k_1, \dots, k_r$ largest elements among $x_1, \dots, x_R \sim P$.*

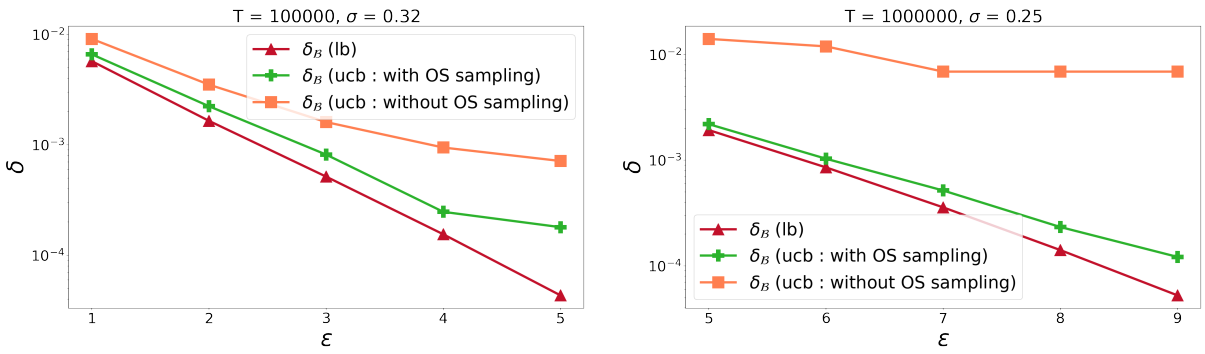

Figure 4: Upper confidence bounds on $\delta_{\mathcal{B}}(\varepsilon)$ against various values of ε for two settings of T and σ, with and without order statistics sampling for roughly the same running time complexity. Since order statistics sampling offers a significant speed up, it affords a larger sample complexity. Additionally, lower bounds on $\delta_{\mathcal{B}}(\varepsilon)$ are included.

Proof. We prove the statement via induction on r. When $r = 1$, we have from Fact B.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 $(y^{(k_1)}, \ldots, y^{(k_{r-1})})$ are jointly distributed as the $(k_1, \ldots, k_{r-1})$ 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 $(y^{(k_1)}, \ldots, y^{(k_{r-1})})$ 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 B.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. □

Numerical evaluation. To demonstrate the usefulness of our order statistics sampling method, in Figure 4, we plot the upper confidence bound on $\delta_{\mathcal{B}}(\varepsilon)$ as obtained via Algorithm 4 as is (i.e., without order statistics sampling) and with order statistics sampling Algorithm 6 (i.e., with an upper bound on the loss function) along the lower bound obtained via (6). The sub-figures in Figure 4 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 $(1, 2, \ldots, 400, 410, \ldots, 1000, 1100, \ldots, 10\,000, 11\,000, \ldots, 50\,000)$ (a total of 590 orders) were used. 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 $m = 1000$ samples, whereas, the estimates with order statistics used $m = 300,000$ samples, using the order statistics of $(1, 2, \ldots, 300, 310, \ldots, 1000, 1100, \ldots, 10\,000, 11\,000, \ldots, 100\,000, 110\,000, \ldots, 500\,000)$ (a total of 590 orders) were used. 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 machine; these figures are offered as a rough guide only. 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.

B.3 Combining Importance and Order Statistics Sampling

We sketch how the techniques of importance sampling and order statistics sampling can be used together.

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 $E_\varepsilon := \{x :\in \mathbb{R}^T : \max_{t \in [T]} x_t \leq C_\varepsilon\}$ for $C_\varepsilon := \frac{1}{2} - \varepsilon\sigma^2$. 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 6 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 $x_1, \ldots, x_T \leq C_\varepsilon$, via

Algorithm 11 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

Proposition B.4. And finally, we use Algorithm 5 where we replace $L_{Q_{\mathcal{B}} \parallel P_{\mathcal{B}}}(x)$ by an upper bound in terms of the order statistics given as,

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

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 $P_1 = \mathcal{N}(e_1, \sigma^2 I)$ 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 10 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 6 to sample the order statistics $y^{(k_1)}, \ldots, y^{(k_r)}$ with $R = T - 1$ and use the following upper bound on $L_{P_{\mathcal{B}} \parallel Q_{\mathcal{B}}}$:

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

- 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 B.3, and use a small variant of Algorithm 6 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 $L_{P_{\mathcal{B}} \parallel Q_{\mathcal{B}}}(x)$:

$$\log\left(e^{x_1/\sigma^2} + e^{x_2/\sigma^2} + \sum_{i=1}^r (k_{i+1} - k_i) \cdot e^{y^{(k_i)}/\sigma^2}\right) - \log T - \frac{1}{2\sigma^2}.$$

B.4 Privacy Accounting of 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 11, which relies on the simple observation that

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

The order statistics sampling technique can be extended to this case by applying it independently to sample an upper bound on $L_{P \parallel Q}(x^{(i,j)})$ 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.

C 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 1024 and 262 144, and we tune the norm bound $C \in \{1, 5, 10, 15, 30, 50, 100, 500, 1000\}$. 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, and 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 $\delta_{\mathcal{B}}(\varepsilon)$ we consider. In particular, we use maximum batch sizes in $\{1328, 2469, 4681, 9007, 17\,520, 34\,355, 67\,754, 134\,172, 266\,475\}$ for the Criteo pCTR dataset and $\{1320, 2458, 4665, 8984, 17\,488, 34\,309, 67\,687, 134\,071, 266\,317\}$ for the Criteo Search dataset, corresponding to the expected batch sizes of $\{1024, 2048, 4096, 8192, 16\,384, 32\,768, 65\,536, 131\,072, 262\,144\}$.

For the privacy accounting in Figure 2, 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 1024, there are total of $T = 36\,133$ steps. We use the order statistics of $(1, 2, \ldots, 500, 510, 520, \ldots, 1000, 1100, 1200, \ldots, 19\,900)$, which involves a total of 739 orders, which is about 2% of the number of steps.
  - ▷ For expected batch size 8192, there are a total of $T = 4517$ steps. We use the order statistics of $(1, 2, \ldots, 500, 510, 520, \ldots, 1000, 1050, 1100, \ldots, 2950)$, 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 1024, there are total of $T = 12\,497$ steps. We use the order statistics of $(1, 2, \ldots, 500, 510, 520, \ldots, 1000, 1100, 1200, \ldots, 6900)$, which involves a total of 609 orders, which is about 4.8% of the number of steps.
  - ▷ For expected batch size 8192, there are a total of $T = 1563$ 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 $L_{P_{\mathcal{B}} \parallel Q_{\mathcal{B}}}(x)$ (resp. $L_{Q_{\mathcal{B}} \parallel P_{\mathcal{B}}}(x)$) using the order statistics sampling (Algorithm 6), 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.

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

We elaborate on Remark 3.5 showing that $\mathsf{ABLQ}_{\mathcal{B}}$ and $\mathsf{ABLQ}_{\mathcal{P}}$ have incomparable privacy guarantees.

Theorem D.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 D.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.

Fact D.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 D.4 (Converse to Lemma 2.3; (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 D.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 D.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 D.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 2.3, 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 3.4. □

While Theorem 3.4 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 D.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.