Differentially Private Quantiles

Jennifer Gillenwater Matthew Joseph Alex Kulesza

Abstract

Quantiles are often used for summarizing and understanding data. If that data is sensitive, it may be necessary to compute quantiles in a way that is differentially private, providing theoretical guarantees that the result does not reveal private information. However, when multiple quantiles are needed, existing differentially private algorithms fare poorly: they either compute quantiles individually, splitting the privacy budget, or summarize the entire distribution, wasting effort. In either case the result is reduced accuracy. In this work we propose an instance of the exponential mechanism that simultaneously estimates exactly m quantiles from n data points while guaranteeing differential privacy. The utility function is carefully structured to allow for an efficient implementation that returns estimates of all m quantiles in time $O(mn\log(n)+m^2n)$. Experiments show that our method significantly outperforms the current state of the art on both real and synthetic data while remaining efficient enough to be practical.

1. Introduction

Quantiles are a widespread method for understanding real-world data, with example applications ranging from income (Semega et al., 2020) to birth weight (CDC, 2001) to standardized test scores (ETS, 2020). At the same time, the individuals contributing data may require that these quantiles not reveal too much information about individual contributions. As a toy example, suppose that an individual joins a company that has exactly two salaries, and half of current employees have one salary and half have another. In this case, publishing the exact median company salary will reveal the new employee's salary.

Differential privacy (Dwork et al., 2006) offers a solution to this problem. Informally, the distribution over a differentially private algorithm's outputs must be relatively insensitive to the input of any single data contributor. Returning to the salary example, a differentially private method for computing the median company salary would have similar-looking output distributions regardless of which salary the new employee receives. The resulting uncertainty about any single contributor's data makes the algorithm "private".

In this work, we study differentially private estimation of user-specified quantiles $q_1, \ldots, q_m \in [0,1]$ for a one-dimensional dataset X of size n. The output quantile estimates consist of m values, which we denote $o_1, \ldots, o_m$. Ideally, the o_j are as close to the dataset's actual quantiles as possible. For example, if $q_j = 0.5$, then our goal is to output o_j close to the median of X.

Several algorithms for computing a single differentially private quantile exist (see Section 5). These naturally extend to multiple quantiles using composition. Basic composition says that, if we estimate each of m quantiles via an $\frac{\varepsilon}{m}$-differentially private algorithm, then we will obtain ε-differential privacy overall for the set of m quantiles. However, the cost of this generality is the smaller and more restrictive privacy budget $\frac{\varepsilon}{m}$ (or roughly $\frac{\varepsilon}{\sqrt{m}}$ for "advanced" composition). As a result, this approach yields significantly less accurate outcomes as m grows. This is unfortunate, as many applications rely on multiple quantiles: returning to the opening paragraph, the income statistics use $m = 4$ (quintiles), the birth weight statistics use $m = 9$ (deciles), and the test score statistics use $m > 30$. Alternatively, there exist methods for computing a differentially private summary of the entire distribution from which any quantile can subsequently be estimated (see Section 1.2). However, unless m is very large, such summaries will usually contain more information than needed, reducing accuracy.

1.1. Contributions

1. We give an instantiation of the exponential mechanism (McSherry & Talwar, 2007), *JointExp*, that produces an ε-differentially private collection of m quantile estimates in a single invocation (Section 3.1). This mechanism uses a utility function that has sensitivity 2 no matter how many quantiles are requested, and does not need to divide ε based on the number of quantiles.

Equal contributions, all authors at Google Research New York. Correspondence to: Jennifer Gillenwater <jengi@google.com>, Matthew Joseph <mtjoseph@google.com>, Alex Kulesza <kulesza@google.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

2. We provide a dynamic program, related to algorithms used for inference in graphical models, to implement *JointExp* in time $O(mn\log(n) + m^2 n)$ (Section 3.2, Section 3.3). This significantly improves on naive sampling, which requires time $O(n^m)$.

3. We experimentally evaluate *JointExp* and find that it obtains much better accuracy than the existing state-of-the-art while remaining efficient enough to be practical for moderate dataset sizes (Section 5).

1.2. Related Work

Discussion of single-quantile estimation algorithms appears in Section 5. At the other end of the spectrum, one can use private CDF estimation or private threshold release to estimate arbitrarily many quantiles. These approaches avoid splitting ε as m grows but suffer from the need to set hyperparameters depending on the discretization of the domain and assumptions about the data distribution. Moreover, the best known algorithms for threshold release rely on several reductions that limit their practicality (Bun et al., 2015; Kaplan et al., 2020). A common tree-based approach to CDF estimation is included in our experiments.

Our algorithm relies on dynamic programming to sample from the exponential mechanism. Blocki et al. (2016) studied how to release the counts (but not identities) of items in a dataset by constructing a relaxation of the exponential mechanism and sampling from it using dynamic programming. However, their utility function more simply decomposes into individual terms without pairwise interactions, and it is not clear how this method can be applied to quantiles.

Finally, our dynamic program for sampling from *JointExp*'s exponential mechanism is related to inference algorithms for graphical models. Several papers have studied differential privacy with graphical models. However, this has typically meant studying private versions of graphical modeling tasks (Williams & McSherry, 2010; Bernstein et al., 2017) or using graphical models as a step in private algorithms (Mckenna et al., 2019). Our paper departs from that past work in that its dynamic program, while related to the forward-backward algorithm, does not have any conceptual dependence on graphical models themselves.

2. Preliminaries

We view databases X, X' as multisets of elements from some data domain $\mathcal{X}$ where each individual contributes at most one element to the database. To reason about databases that are "close", differential privacy uses *neighbors*.

Definition 1. *Databases X and $X' \in \mathcal{X}^n$ are* neighbors, *denoted $X \sim X'$, if they differ in at most one element.*

Note that we use the *swap* definition of differential privacy; in contrast, the *add-remove* definition allows the addition or removal (rather than exchange) of one element between neighboring databases. We do this for consistent evaluation against the smooth-sensitivity framework (see Appendix E), which also uses swap differential privacy. However, we emphasize that our algorithm *JointExp* easily adapts to the add-remove framework (in fact, its sensitivity is lower under add-remove privacy).

With the notion of neighboring databases in hand, we can now define differential privacy.

Definition 2 (Dwork et al. (2006))**.** *A randomized algorithm $\mathcal{A}\colon \mathcal{X}^* \to \mathcal{Y}$ is (ε, δ)-differentially private if, for every pair of neighboring databases X, X' and every output subset $Y \subseteq \mathcal{Y}$,*

$$\mathbb{P}_{\mathcal{A}}\left[A(X) \in Y\right] \leq e^{\varepsilon}\mathbb{P}_{\mathcal{A}}\left[A(X') \in Y\right] + \delta.$$

When $\delta > 0$, we say $\mathcal{A}$ satisfies approximate *differential privacy. If $\delta = 0$, we say $\mathcal{A}$ satisfies* pure *differential privacy, and shorthand this as ε-differential privacy (or ε-DP).*

A key benefit of differential privacy is composition: an algorithm that relies on differentially private subroutines inherits an overall privacy guarantee by simply adding up the privacy guarantees of its components.

Lemma 1 (Dwork et al. (2006))**.** *Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be k algorithms that respectively satisfy $(\varepsilon_1, \delta_1)$-, $\ldots$, $(\varepsilon_k, \delta_k)$-differential privacy. Then running $\mathcal{A}_1, \ldots, \mathcal{A}_k$ satisfies $\left(\sum_{i=1}^k \varepsilon_i, \sum_{i=1}^k \delta_i\right)$-differential privacy.*

We will use composition (or its "advanced" variants) when evaluating methods that estimate a set of m quantiles by estimating each quantile individually. By Lemma 1, to achieve overall ε-DP, it suffices to estimate each quantile under $\frac{\varepsilon}{m}$-DP. However, since our algorithm *JointExp* estimates all quantiles in one invocation, it does not use composition.

We will also rely on the exponential mechanism, a common building block for differentially private algorithms.

Definition 3 (McSherry & Talwar (2007); Dwork & Roth (2014))**.** *Given utility function $u\colon \mathcal{X}^* \times O \to \mathbb{R}$ mapping (database, output) pairs to real-valued scores with L_1 sensitivity $\Delta_u = \max_{X \sim X', o \in O} |u(X, o) - u(X', o)|$, the exponential mechanism M has output distribution*

$$\mathbb{P}_M\left[M(X) = o\right] \propto \exp\left(\frac{\varepsilon u(X, o)}{2\Delta_u}\right),$$

where $\propto$ elides the normalization factor.

The exponential mechanism thus prioritizes a database's higher-utility outputs while remaining private.

Lemma 2 (McSherry & Talwar (2007))**.** *The mechanism described in Definition 3 is ε-DP.*

The above material suffices to understand the bulk of our algorithm, *JointExp*. The algorithms used for our experimental comparisons will also require some understanding of smooth sensitivity and concentrated differential privacy, but since these concepts will be relevant only as points of experimental comparison, we discuss them in Section 5.

3. *JointExp*

This section provides an exposition of our quantiles algorithm, *JointExp*. Recall that our goal is to take as input quantiles $q_1 < q_2 < \ldots < q_m \in [0,1]$ and database X and output quantile estimates $o_1, \ldots, o_m$ such that, for each $j \in [m]$, $\mathbb{P}_{x \sim_U X}[x \leq o_j] \approx q_j$.

In Section 3.1, we start with an instance of the exponential mechanism whose continuous output space makes sampling impractical. In Section 3.2, we construct a mechanism with the same output distribution (and, importantly, the same privacy guarantees) and a bounded but inefficient sampling procedure. Finally, in Section 3.3 we modify our sampling procedure once more to produce an equivalent and polynomial time method, which we call *JointExp*.

3.1. Initial Solution

We start by formulating an instance of the exponential mechanism for our quantiles setting. First, we will require the algorithm user to input a lower bound a and upper bound b for the data domain.[1] We assume that all $x \in X$ are in $[a,b]$; if this is not the case initially, then we clamp any outside points to $[a,b]$. The output space is $O_{\nearrow} = \{(o_1, \ldots, o_m) \mid a \leq o_1 \leq \cdots \leq o_m \leq b\}$, the set of sequences of m nondecreasing values from $[a,b]$. For a given $o = (o_1, \ldots, o_m)$, the utility function will compare the number of points in each proposed quantile interval $[o_{j-1}, o_j)$ to the expected number of points in the correct quantile interval. We denote the number of data points between adjacent quantiles q_{j-1} and q_j by $n_j = (q_j - q_{j-1})n$. We fix $q_0 = 0$ and $q_{m+1} = 1$, so that $n_1 = q_1 n$ and $n_{m+1} = (1 - q_m)n$. We also denote the number of data points from X between any two values u and v by

$$n(u,v) = |\{x \in X \mid u \leq x < v\}|.$$

We can now define our utility function

$$u_Q(X,o) = -\sum_{j \in [m+1]} |n(o_{j-1}, o_j) - n_j|,$$

where we fix $o_0 = a$ and $o_{m+1} = b + 1$ (setting o_{m+1} to a value strictly larger than b simply ensures that points equal to b are counted in the final term of the sum). u_Q thus assigns highest utility to the true quantile values and lower utility to estimates that are far from the true quantile values.

Lemma 3. *u_Q has L_1 sensitivity $\Delta_{u_Q} = 2$.*

Proof. Fix an output o. Let X and X' be neighboring databases. Since we use swap differential privacy, $|X| = |X'|$, so $u_Q(X,o)$ and $u_Q(X',o)$ only differ in their $n(\cdot,\cdot)$ (respectively denoted $n'(\cdot,\cdot)$ for X'). Since X and X' are neighbors, there are at most two intervals (o_{j-1}, o_j) and $(o_{j'-1}, o_{j'})$ on which n and n' differ, each by at most one. Thus $|u_Q(X,o) - u_Q(X',o)| \leq 2$. □

For add-remove privacy, the sensitivity is slightly lower at $\Delta_{u_Q} = 2[1 - \min_{j \in [m+1]}(q_j - q_{j-1})]$. A full proof of this and other results appears in Appendix A.

The corresponding mechanism M_Q has output density

$$f_Q(o) \propto \cdot \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_Q(X,o)\right). \tag{1}$$

Since this is an instantiation of the exponential mechanism, we can apply Lemma 2 to get:

Lemma 4. *The mechanism M_Q defined by output density f_Q satisfies ε-differential privacy.*

However, as is typically a drawback of the exponential mechanism, it is not clear how to efficiently sample from this distribution, which is defined over a continuous m-dimensional space. The following sections address this issue. Since the output distribution itself remains fixed through these sampling procedure changes, these improvements will preserve the privacy guarantee of Lemma 4. The remaining proofs will therefore focus on verifying that subsequent sampling procedures still sample according to Eq. 1.

3.2. Finite Sampling Improvement

In this section, we describe how to sample from the continuous distribution defined by M_Q by first sampling from an intermediate discrete distribution. This is similar to the single-quantile sampling technique given by Smith (2011) (see their Algorithm 2). The basic idea is that we split the sampling process into three steps:

1. Sample m intervals from the set of intervals between data points.
2. Take a uniform random sample from each of the m sampled intervals.
3. Output the samples in increasing order.

This will require some additional notation. Denote the elements of X in nondecreasing order by $x_1 \leq \cdots \leq x_n$, fix

[1] Lower and upper bounds are also necessary for the private quantile algorithms that we compare to in our experiments. We find that choosing loose bounds a and b does not greatly affect utility (see experiments in Section 5).

$x_0 = a$ and $x_{n+1} = b$, and let $I = \{0, \dots, n\}$, where we associate $i \in I$ with the interval between points x_i and x_{i+1}. Define $S_{\nearrow}$ to be the set of nondecreasing sequences of m intervals,

$$S_{\nearrow} = \{(i_1, \dots, i_m) \mid i_1, \dots, i_m \in I, i_1 \le \cdots \le i_m\}.$$

$S_{\nearrow}$ will be the discrete output space for the first sampling step above. We can define a utility function $u_{Q'}$ on $s = (i_1, \dots, i_m) \in S_{\nearrow}$ by slightly modifying u_Q:

$$u_{Q'}(X, s) = -\sum_{j \in [m+1]} |(i_j - i_{j-1}) - n_j|,$$

where we fix $i_0 = 0$ and $i_{m+1} = n$.

In order to reproduce M_Q from Section 3.1, our sequence sampler will also need to weight each sequence s by the total measure of the outputs $o \in O_{\nearrow}$ that can be sampled from s in the second step. This is nontrivial due to the ordering constraint on o: if an interval appears twice in s, the measure of corresponding outputs must be halved to account for the fact that the two corresponding samples in the second step can appear in either order, but will be mapped to a fixed increasing order in the third step. In general, if an interval appears k times, the measure must be scaled by a factor of $1/k!$, the volume of the standard k-simplex. We account for this by dividing by the scale function

$$\gamma(s) = \prod_{i \in I} \mathrm{count}_s(i)!\,,$$

where $\mathrm{count}_s(i)$ is the number of times i appears in s and we take $0! = 1$.

The mechanism $M_{Q'}$ is now defined as follows:

1. Draw $s = (i_1, \dots, i_m)$ according to
$$\mathbb{P}_{M_{Q'}}[s] \propto \exp\left(\frac{\varepsilon \cdot u_{Q'}(X, s)}{2\Delta_{u_Q}}\right) \cdot \frac{\prod_{j=1}^{m}(x_{i_j+1} - x_{i_j})}{\gamma(s)}.$$
2. For $j \in [m]$, draw o_j uniformly at random from $[x_{i_j}, x_{i_j+1})$.
3. Output $o_1, \dots, o_m$ in increasing order.

It remains to verify that $M_{Q'}$ actually matches M_Q.

Lemma 5. *$M_{Q'}$ has the same output distribution as M_Q.*

Proof Sketch (see Appendix A for full proof). Given potential outputs o and o', if the corresponding quantile estimates fall into the same intervals between data points in X, then the counts $n(\cdot, \cdot)$ are unchanged and $u_Q(X, o) = u_Q(X, o')$. Since u_Q is constant over intervals between data points, it is equivalent to sample those intervals and then sample points uniformly at random from the chosen intervals. The only complication is accounting for the γ scaling introduced by repeated intervals. □

The benefit of $M_{Q'}$ over M_Q is that the first step samples from a finite space, and the second sampling step is simply uniform sampling. However, the size of the space for the first step is still $O(n^m)$, which remains impractical for all but the smallest datasets. In the next section, we develop a dynamic programming algorithm that allows us to sample from $\mathbb{P}_{M_{Q'}}$ in time $O(mn \log(n) + m^2 n)$.

3.3. Polynomial Sampling Improvement

Notice that the bulk of our probability distribution over sequences $s = (i_1, \dots, i_m)$ can be decomposed as a product of scores, where each score depends only on adjacent intervals i_{j-1} and i_j. In particular,

$$\mathbb{P}_{M_{Q'}}[s] \propto \frac{1}{\gamma(s)} \prod_{j \in [m+1]} \phi(i_{j-1}, i_j, j) \prod_{j \in [m]} \tau(i_j)\,,$$

where for $i \le i'$ and $j \in [m+1]$ we define

$$\phi(i, i', j) = \exp\left(-\frac{\varepsilon}{2\Delta_{u_Q}} |(i' - i) - n_j|\right)$$
$$\tau(i) = x_{i+1} - x_i.$$

For $i > i'$ and any j, $\phi(i, i', j) = 0$. Fig. 1 illustrates this structure graphically, suggesting a dynamic programming algorithm similar to the "forward-backward" algorithm from the graphical models literature (see, e.g., Chapter 15 of Russell & Norvig (2010)).

Unfortunately, $\gamma(s)$ does not factor in the same way. However, it has its own special structure: since s is required to be nondecreasing, $\gamma(s)$ decomposes over contiguous constant *subsequences* of s. We will use this to design an efficient dynamic programming algorithm for sampling $\mathbb{P}_{M_{Q'}}$.

Define the function $\alpha\colon [m] \times I \times [m] \to \mathbb{R}$ so that $\alpha(j, i, k)$ is the total unnormalized probability mass for prefix sequences of length j that end with exactly k copies of the interval i. For all $i \in I$, let $\alpha(1, i, 1) = \phi(0, i, 1)\tau(1)$ and $\alpha(1, i, k) = 0$ for $k > 1$. Now, for $j = 2, \dots, m$, we have the following recursion for all $i \in I$:

$$\alpha(j, i, 1) = \tau(i) \sum_{i' < i} \phi(i', i, j) \sum_{k < j} \alpha(j - 1, i', k)$$
$$\alpha(j, i, k > 1) = \tau(i) \cdot \phi(i, i, j) \cdot \alpha(j - 1, i, k - 1)/k$$

Intuitively, if the sequence ends with a single i, we need to sum over all possible preceding intervals i', which could have been repeated up to $k < j$ times. On the other hand, if the sequence ends with more than one i, we know that the preceding interval was also i, and we simply divide by k to account for the scale function γ.

Having computed $\alpha(\cdot, \cdot, \cdot)$, we can now use these quantities to sample in the reverse direction as follows. First, draw a

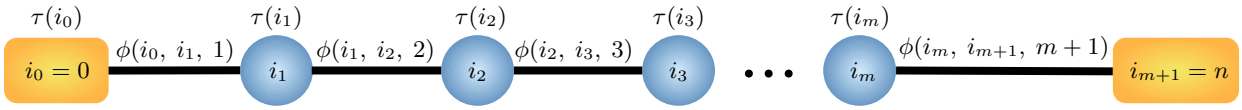

Figure 1. Illustration of pairwise dependencies for interval sequence $s = (i_1, \ldots, i_m)$.

pair $(i, k) \propto \alpha(m, i, k)\phi(i, n, m+1)$ (the ϕ term accounts for the final edge in the graph; see Appendix A for details). This determines that the last k sampled intervals are equal to i. We can then draw another pair $(i' < i, k') \propto \alpha(m-k, i', k')\phi(i', i, m-k+1)$, which determines that the last k' remaining intervals in the sequence are i', and so on until we have a complete sample.

We will verify that this procedure actually samples from the correct distribution in the proof of Theorem 1. For now, we turn to an optimized version of this procedure, presented in Algorithm 1. The main optimization leverages the structure of $\phi(\cdot, \cdot, j)$: fixing j, $\phi(i, i', j)$ depends only on $i' - i$. $\phi(\cdot, \cdot, j)$ is therefore a matrix with constant diagonals, i.e. a Toeplitz matrix. A key benefit of $n \times n$ Toeplitz matrices is that matrix-vector multiplication can be implemented in time $O(n \log(n))$ using the Fast Fourier Transform (see, e.g., (Bindel, 2019)). This becomes useful to us once we rewrite the computation of $\alpha(j, \cdot, \cdot)$ using

$$\hat{\alpha}(j-1, \cdot) = \sum_{k<j} \alpha(j-1, \cdot, k)$$
$$\alpha(j, \cdot, 1) = \tau(\cdot) \times \left(\phi(\cdot, \cdot, j)^T \hat{\alpha}(j-1, \cdot)\right)$$

where $\times$ denotes element-wise product. This reduces each computation of $\alpha(j, \cdot, 1)$ in Line 9 of Algorithm 1 to time $O(n \log(n))$ and space $O(n)$.

In total, we spend time $O(m^2 n)$ computing $\hat{\alpha}(\cdot, \cdot)$, time $O(mn \log(n))$ computing $\alpha(\cdot, \cdot, 1)$, and time $O(m^2)$ computing $\alpha(\cdot, \cdot, k)$ for $k > 1$. The result is overall time $O(mn \log(n) + m^2 n)$. The space analysis essentially reduces to the space needed to store α while computing ϕ as needed. Details appear in the proof of Theorem 1.

Theorem 1. *JointExp satisfies ε-differential privacy, takes time $O(mn \log(n) + m^2 n)$, and uses space $O(m^2 n)$.*

Numerical improvements. Note that the quantities involved in computing ϕ and α may be quite small, so we implement *JointExp* using logarithmic quantities to avoid underflow errors in our experiments. This is a common trick and is a numerical rather than algorithmic change, but for completeness we include its details in Appendix B. After computing these quantities, to avoid underflow in our final sampling steps, we use a "racing" sampling method that was previously developed for single-quantile exponential mechanisms. Since this is again a numerical improvement, details appear in Appendix C.

Algorithm 1 Pseudocode for *JointExp*

1: **Input:** sorted $X = (x_1 \le \ldots \le x_n)$ clamped to data range $[a, b]$, quantiles $q_1, \ldots, q_m$, privacy parameter ε
2: Set $x_0 = a$, $x_{n+1} = b$, and $\Delta_{u_Q} = 2$
3: Set $I = \{0, \ldots, n\}$, $i_0 = 0$, and $i_{m+1} = n$
4: **for** $i \in I$ **do**
5: Set $\alpha(1, i, 1) = \phi(0, i, 1)\tau(1)$
6: $= \exp\left(-\frac{\varepsilon}{2\Delta_{u_Q}}|1 - n_1|\right) \cdot (x_1 - x_0)$
7: **for** $j = 2, \ldots, m$ **do**
8: **for** $i \in I$ **do**
9: Set $\hat{\alpha}(j-1, i) = \sum_{k<j} \alpha(j-1, i, k)$
10: Set $\alpha(j, \cdot, 1) = \tau(\cdot) \times \left(\phi(\cdot, \cdot, j)^T \hat{\alpha}(j-1, \cdot)\right)$
11: **for** $k = 2, \ldots, j$ **do**
12: Set $\alpha(j, i, k) = \tau(i)\phi(i, i, j)\alpha(j-1, i, k-1)/k$
13: Sample $(i, k) \propto \alpha(m, i, k)\phi(i, n, m+1)$
14: Set $i_{m-k+1}, \ldots, i_m = i$, and $j = m - k$
15: **while** $j > 0$ **do**
16: Sample $(i, k) \propto \alpha(j, i, k)\phi(i, i_{j+1}, j+1)\tau(i_{j+1})$
17: Set $i_{j-k+1}, \ldots, i_j = i$, and $j = j - k$
18: Output uniform samples $\{o_j \sim_U [x_{i_j}, x_{i_j+1})\}_{j=1}^m$ in increasing order

Connection to graphical models. As mentioned above, the dynamic program in Algorithm 1 is similar to the forward-backward algorithm from the graphical models literature, modulo accounting for $\gamma(s)$. In graphical models, it is often necessary to compute the probability of a sequence of hidden states. This requires normalizing by a sum of probabilities of sequences, and, naively, this sum has an exponential number of terms. However, when probabilities decompose into products of score functions of adjacent states, the forward-backward algorithm makes the process efficient. The extra $\gamma(s)$ term makes our sampling process more complex in a way that is similar to semi-Markov models (Yu, 2010). In graphical model terms, γ can be thought of as a prior that discourages repeats: $p_{\text{prior}}(s) \propto 1/\gamma(s)$. This prior can also be written as a product of n Poisson distributions, each with parameter $\lambda = 1$.

4. Accuracy Intuition

JointExp applies the exponential mechanism once to output m quantiles. The closest competitor algorithms also apply the exponential mechanism but use m invocations to produce m quantiles. To build intuition for why the for-

mer approach achieves better utility, we recall the standard accuracy guarantee for the exponential mechanism:

Lemma 6 (McSherry & Talwar (2007)). *Let M be an ε-DP instance of the exponential mechanism having score function u with sensitivity Δ_u and output space $\mathcal{Y}$. Then for database X, with probability at least $1-\beta$, M produces output y such that*

$$u(X,y) \geq \max_{y^* \in \mathcal{Y}} u(X,y^*) - \frac{2\Delta_u \log(|\mathcal{Y}|/\beta)}{\varepsilon}.$$

For simplicity, suppose we have uniform data where all interval widths $x_{i+1} - x_i$ are identical. As shown by the experiments in the next section, this is not necessary for *JointExp* to obtain good utility, but we assume it for easier intuition. Then (modulo the minor term $\gamma(s)$ that accounts for rare repeated intervals in the output), $\mathbb{P}_{M_{Q'}}[s] \propto \exp\left(\frac{\varepsilon \cdot u_{Q'}(X,s)}{2\Delta_{u_Q}}\right)$. This means that *JointExp*'s process of sampling intervals draws from a distribution whose shape is identical to an exponential mechanism with utility function $u_{Q'}$, but mismatched sensitivity term $\Delta_{u_Q} = 2$. Since the proof of Lemma 6 does not rely on the utility function matching the sensitivity term, we can still apply it to determine the accuracy of this interval sampling procedure. The output space $\mathcal{Y}$ for *JointExp*'s interval-sampling has size $|S_{\nearrow}| \leq n^m$, so we expect to sample intervals yielding quantiles that in total misclassify $O(m\log(n)/\varepsilon)$ points.

In contrast, m invocations of a single-quantile exponential mechanism requires each invocation to satisfy roughly $\varepsilon_i = \varepsilon/\sqrt{m}$-DP (advanced composition). Because each invocation uses an output space of size $O(n)$, the total error guarantee via Lemma 6 scales like $O(m\log(n)/\varepsilon_i)$. Since $m/\varepsilon_i = \omega(m)/\varepsilon$ for even the best known composition bounds for the exponential mechanism (Dong et al., 2020), these approaches incur error with a superlinear dependence on m. This contrasts with *JointExp*'s error, which has only a linear dependence on m.

5. Experiments

We now empirically evaluate *JointExp* against three alternatives: *AppIndExp*, *CSmooth*, and *AggTree*. Discussion of some omitted alternatives appears in Appendix D. All experiment code is publicly available (Google, 2021).

5.1. Comparison Algorithms

***AppIndExp*:** Our first comparison algorithm *AppIndExp* uses independent applications of the exponential mechanism. Smith (2011) introduced the basic *IndExp* algorithm for estimating one quantile, and it has since been incorporated into the SmartNoise (SmartNoise, 2020) and IBM (IBM, 2019) differential privacy libraries. *IndExp* thus gives us a meaningful baseline for a real-world approach.

IndExp uses the exponential mechanism to estimate a single quantile q via the utility function $u(X,o) = ||\{x \in X \mid x \leq o\}| - qn|$.

Lemma 7. *u defined above has L_1 sensitivity $\Delta_u = 1$.*

Proof. Consider swapping $x \in X$ for x', and fix some o. If $x, x' \leq o$ or $x, x' > o$, then $u(X,o) = u(X',o)$. If exactly one of x or x' is $\leq o$, then $|u(X,o) - u(X',o)| = 1$. □

IndExp takes user-provided data bounds a and b and runs on $X^+ = X \cup \{a,b\}$. After sorting X^+ into intervals of adjacent data points $I_0, \ldots, I_n$, *IndExp* selects an interval I_j with probability proportional to

$$\text{score}(X, I_j) = \exp(-\varepsilon|j - qn|/2) \cdot |I_j|$$

and randomly samples the final quantile estimate from I_j.

To estimate m quantiles with *AppIndExp*, we call *IndExp* m times with ε computed using the exponential mechanism's nonadaptive composition guarantee (Dong et al., 2020). Details appear in Appendix E, but we note that this is the tightest known composition analysis for the exponential mechanism. Since our experiments use $n = 1000$ data points, we always use $\delta = 10^{-6}$ in accordance with the recommendation that $\delta \ll \frac{1}{n}$ (see the discussions around the definition of differential privacy from Dwork & Roth (2014) and Vadhan (2017)).

***CSmooth*:** Our second comparison algorithm is *CSmooth*, which combines the *smooth sensitivity* framework introduced by Nissim et al. (2007) with concentrated differential privacy (CDP) (Dwork & Rothblum, 2016; Bun & Steinke, 2016). The basic idea of smooth sensitivity is to circumvent global sensitivity by instead using a smooth analogue of local sensitivity. This is useful for problems where the global sensitivity is large only for "bad" datasets.

Definition 4. *For function $f : X^n \to \mathbb{R}$, the* local sensitivity *$\Delta_f(X)$ of f for dataset X is $\max_{X' | X \sim X'} |f(X) - f(X')|$.*

Recall that global sensitivity is defined over all possible pairs of datasets. In contrast, local sensitivity is also parameterized by a fixed dataset X and defined only over neighbors of X. It is therefore possible that $\Delta_f(X) \ll \Delta_f$. For example, if M is the median function and we set $X = [-100, 100]$, then $\Delta_M(\{-1,0,1\}) = 1$ while $\Delta_M(\{-100,0,100\}) = 100$. However, this also shows that local sensitivity itself reveals information about the dataset. The insight of Nissim et al. (2007) is that it is possible to achieve differential privacy and take advantage of lower local sensitivity by adding noise calibrated to a "smooth" approximation of $\Delta_f(X)$.

Definition 5 (Nissim et al. (2007)). *For $t > 0$, the t-*smooth sensitivity *of f on database X of n points is*

$$\mathcal{S}_f^t(X) = \max_{X' \in \mathcal{X}^n} e^{-t \cdot d(X,X')} \cdot \Delta_f(X').$$

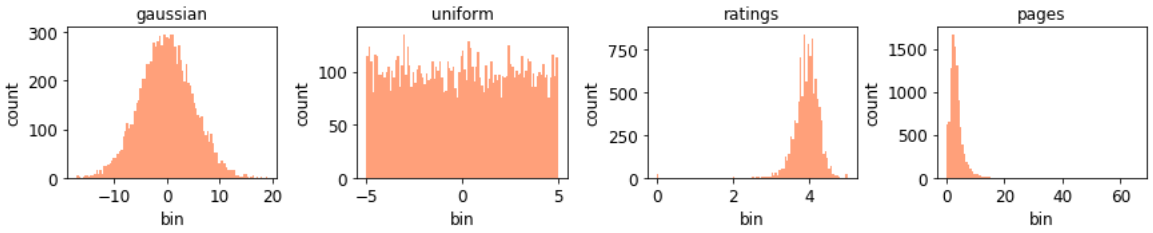

Figure 2. Histograms for the four datasets. Each plot uses 10,000 random samples, and the data range is divided into 100 equal-width bins.

Details for computing the median's smooth sensitivity appear in Appendix E. We now turn to the CDP portion of *CSmooth*. CDP is a variant of differential privacy that offers comparable privacy guarantees with often tighter privacy analyses. Bun & Steinke (2019) showed how to combine CDP with the smooth sensitivity framework. Our experiments use the Laplace Log-Normal noise distribution, which achieved the strongest accuracy results in the experiments of Bun & Steinke (2019).

One complication of *CSmooth* is the need to select several parameters to specify the noise distribution. We tuned these parameters on data from $N(0,1)$ to give *CSmooth* the strongest utility possible without granting it distribution-specific advantages. Details appear in Appendix E. To compare the *JointExp*'s pure DP guarantee to *CSmooth*'s CDP guarantee, we use the following lemma:

Lemma 8 (Proposition 1.4 (Bun & Steinke, 2016))**.** *If an algorithm is ε-DP, then it is also $\frac{\varepsilon^2}{2}$-CDP.*

We thus evaluate our ε-DP algorithm *JointExp* against an $\frac{\varepsilon^2}{2}$-CDP *CSmooth*. This comparison favors *CSmooth*: recalling our requirement that approximate DP algorithms have $\delta \leq 10^{-6}$, the best known generic conversion from CDP to approximate DP only says that a $\frac{1}{2}$-CDP algorithm is $(\varepsilon, 10^{-6})$-DP for $\varepsilon \geq 5.76$ (Proposition 1.3, (Bun & Steinke, 2016)). A more detailed discussion of DP and CDP appears in Section 4 of the work of Canonne et al. (2020).

As with *AppIndExp*, to estimate m quantiles with *CSmooth*, we call it m times with an appropriately reduced privacy parameter. This time, we use CDP's composition guarantee:

Lemma 9 (Proposition 1.7 (Bun & Steinke, 2016))**.** *The composition of k ρ-CDP algorithms is $k\rho$-CDP.*

From Lemma 8 the overall desired privacy guarantee is $\frac{\varepsilon^2}{2}$-CDP, so we use $\varepsilon' = \frac{\varepsilon}{\sqrt{m}}$ in each call.

***AggTree*:** The final comparison algorithm, *AggTree*, implements the tree-based counting algorithm (Dwork et al., 2010; Chan et al., 2011) for CDF estimation. This ε-DP algorithm produces a data structure that yields arbitrarily many quantile estimates. Informally, *AggTree* splits the data domain into buckets and then builds a tree with branching factor b and height h where each leaf corresponds to a bucket. Each node of the tree has a count, and each data point increments the count of h nodes. It therefore suffices to initialize each node with $\mathsf{Lap}\,(h/\varepsilon)$ noise to guarantee ε-DP for the overall data structure, and the data structure now supports arbitrary range count queries. A more detailed exposition appears in the work of Kamath & Ullman (2020). As with *CSmooth*, our experiments tune the hyperparameters b and h on $N(0,1)$ data. We also use the aggregation technique described by Honaker (2015), which combines counts at different nodes to produce more accurate estimates.

5.2. Data Description

We evaluate our four algorithms on four datasets: synthetic Gaussian data from $N(0,5)$, synthetic uniform data from $U(-5,5)$, and real collections of book ratings and page counts from Goodreads (Soumik, 2019) (Figure 2).

5.3. Accuracy Experiments

Our error metric is the number of "missed points": for each desired quantile q_j, we take the true quantile estimate o_j and the private estimate $\hat{o}_j$, compute the number of data points between o_j and $\hat{o}_j$, and sum these counts across all m quantiles. For each dataset, we compare the number of missed points for all five algorithms as m grows. Additional plots for distance error appear in Appendix E.

In each case, the requested quantiles are evenly spaced. $m = 1$ is median estimation, $m = 2$ requires estimating the 33rd and 67th percentiles, and so on. We average scores across 20 trials of 1000 random samples. For every experiment, we take $[-100, 100]$ as the (loose) user-provided data range. For the Goodreads page numbers dataset, we also divide each value by 100 to scale the values to $[-100, 100]$. Experiments for $\varepsilon = 1$ appear in Figure 3.

Across all datasets, a clear effect appears: for a wide range of the number of quantiles m, *JointExp* dominates all other algorithms. At $m = 1$, *JointExp* matches *AppIndExp* and obtains roughly an order of magnitude better error than *CSmooth* or *AggTree*. As m grows, *JointExp* consistently obtains average quantile error roughly 2-3 times smaller

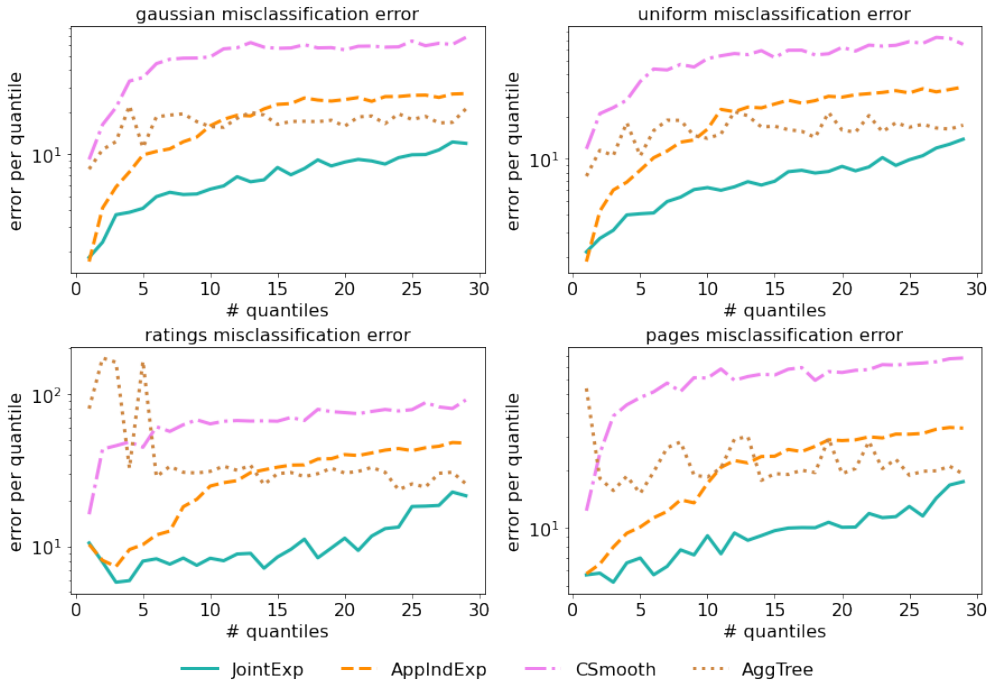

Figure 3. Average # misclassified points per quantile vs. # quantiles, averaged over 50 trials with $\varepsilon = 1$. Note the logarithmic y-axis.

than the closest competitor, until the gap closes around $m = 30$. *JointExp* thus offers both the strongest privacy guarantee and the highest utility for estimating any number of quantiles between $m = 1$ and approximately $m = 30$.

5.4. Time Experiments

We conclude by evaluating the methods by runtime. The number of data points and quantiles are the main determinants of time, so we only include time experiments using Gaussian data. All experiments were run on a machine with two CPU cores and 100GB RAM. As seen in Fig. 6, *JointExp* has time performance roughly in between that of the slowest algorithm, *CSmooth*, and *AppIndExp* or *AggTree*. For estimating $m = 30$ quantiles, *JointExp* takes roughly 1 ms for $n = 1,000$ points and slightly under 1 minute for $n = 1$ million points.

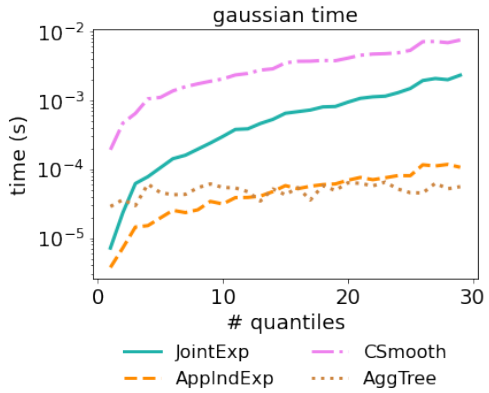

Figure 4. Time vs # quantiles m for $\varepsilon = 1$, averaged across 50 trials of 1,000 samples.

6. Future Directions

In this work we constructed a low-sensitivity exponential mechanism for differentially private quantile estimation and designed a dynamic program to sample from it efficiently. The result is a practical algorithm that achieves much better accuracy than existing methods. A possible direction for future work is exploring other applications of the exponential mechanism where the utility function is low sensitivity and can be decomposed into "local" score functions, as in the pairwise interval terms of ϕ. More precisely, by analogy to the graphical models techniques generally known as belief propagation (Pearl, 1982), any utility function whose outputs have a chain or tree dependency structure should be tractable to sample.

7. Acknowledgements

We thank Thomas Steinke for discussions of concentrated differential privacy; Andrés Muñoz Medina for comments on an early draft of this paper; Uri Stemmer for discussion of the threshold release literature; and Peter Kairouz and Abhradeep Guha Thakurta for discussion of the aggregated tree mechanism.

References

Bernstein, G., McKenna, R., Sun, T., Sheldon, D., Hay, M., and Miklau, G. Differentially private learning of undirected graphical models using collective graphical models. In *International Conference on Machine Learning (ICML)*, 2017.

Bindel, D. Lecture notes for matrix computations. http://www.cs.cornell.edu/courses/cs6210/2019fa/lec/2019-09-04.pdf, 2019.

Blocki, J., Datta, A., and Bonneau, J. Differentially private password frequency lists. In *Network and Distributed System Security (NDSS)*, 2016.

Bun, M. and Steinke, T. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference (TCC)*, 2016.

Bun, M. and Steinke, T. Average-case averages: Private algorithms for smooth sensitivity and mean estimation. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Bun, M., Nissim, K., Stemmer, U., and Vadhan, S. Differentially private release and learning of threshold functions. In *Foundations of Computer Science (FOCS)*, 2015.

Canonne, C., Kamath, G., and Steinke, T. The discrete gaussian for differential privacy. In *Neural Information Processing Systems (NeurIPS)*, 2020.

CDC. Data table of infant weight-for-age charts. https://www.cdc.gov/growthcharts/html_charts/wtageinf.htm, 2001. Accessed: 2021-01-02.

Chan, T.-H. H., Shi, E., and Song, D. Private and continual release of statistics. *Transactions on Information and System Security (TISSEC)*, 2011.

Dong, J., Durfee, D., and Rogers, R. Optimal differential privacy composition for exponential mechanisms and the cost of adaptivity. In *International Conference on Machine Learning (ICML)*, 2020.

Dwork, C. and Lei, J. Differential privacy and robust statistics. In *Symposium on the Theory of Computing (STOC)*, 2009.

Dwork, C. and Roth, A. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 2014.

Dwork, C. and Rothblum, G. N. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887*, 2016.

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

Dwork, C., Naor, M., Pitassi, T., and Rothblum, G. N. Differential privacy under continual observation. In *Symposium on the Theory of Computing (STOC)*, 2010.

ETS. Gre guide to the use of scores. https://www.ets.org/s/gre/pdf/gre_guide.pdf, 2020. Accessed: 2021-01-29.

Google. dp_multiq. https://github.com/google-research/google-research/tree/master/dp_multiq, 2021.

Honaker, J. Efficient use of differentially private binary trees. 2015.

IBM. Ibm differential privacy library. https://github.com/IBM/differential-privacy-library/blob/main/diffprivlib/tools/quantiles.py, 2019. Accessed: 2021-01-05.

Kamath, G. and Ullman, J. A primer on private statistics. *arXiv:2005.00010*, 2020.

Kaplan, H., Ligett, K., Mansour, Y., Naor, M., and Stemmer, U. Privately learning thresholds: Closing the exponential gap. In *Conference on Learning Theory (COLT)*, 2020.

Mckenna, R., Sheldon, D., and Miklau, G. Graphical-model based estimation and inference for differential privacy. In *International Conference on Machine Learning (ICML)*, 2019.

McSherry, F. and Talwar, K. Mechanism design via differential privacy. In *Foundations of Computer Science (FOCS)*, 2007.

Medina, A. M. and Gillenwater, J. Duff: A dataset-distance-based utility function family for the exponential mechanism. *arXiv preprint arXiv:2010.04235*, 2020.

Nissim, K., Raskhodnikova, S., and Smith, A. Smooth sensitivity and sampling in private data analysis. In *Symposium on the Theory of Computing (STOC)*, 2007.

Pearl, J. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Conference on Artificial Intelligence (AAAI)*, 1982.

Russell, S. and Norvig, P. *Artificial Intelligence A Modern Approach*. Pearson Education/Prentice-Hall, third edition, 2010.

scipy. scipy.special.logsumexp. https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.logsumexp.html, 2020. Accessed: 2021-02-08.

Semega, J., Kollar, M., Creamer, J., and Mohanty, A. Income and poverty in the united states: 2018. https://www.census.gov/content/dam/Census/library/publications/2019/demo/p60-266.pdf, 2020. Accessed: 2021-01-02.

SmartNoise. The exponential mechanism for medians. https://github.com/opendifferentialprivacy/smartnoise-core/blob/develop/whitepapers/mechanisms/exponential_median/ExponentialMechForMedian.pdf, 2020. Accessed: 2021-01-05.

Smith, A. Privacy-preserving statistical estimation with optimal convergence rates. In *Symposium on the Theory of Computing (STOC)*, 2011.

Soumik. Goodreads-books dataset. https://www.kaggle.com/jealousleopard/goodreadsbooks, 2019. Accessed: 2020-12-27.

StackOverflow. numerically stable way to multiply log probability matrices in numpy, 2014. Accessed: 2021-02-08.

Tzamos, C., Vlatakis-Gkaragkounis, E.-V., and Zadik, I. Optimal private median estimation under minimal distributional assumptions. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Vadhan, S. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*. Springer, 2017.

Williams, O. and McSherry, F. Probabilistic inference and differential privacy. *Neural Information Processing Systems (NIPS)*, 2010.

Yu, S.-Z. Hidden semi-markov models. *Artificial Intelligence*, 2010.

A. Full Proofs

We start with the add-remove version of the sensitivity analysis for Δ_{u_Q}. We proved the swap version as Lemma 3 in the main body, and this was the focus of the paper. The (slightly more favorable) add-remove version appears below for completeness.

Lemma 10. *In the add-remove model,* $\Delta_{u_Q} = 2[1 - \min_{j\in[m+1]}(q_j - q_{j-1})]$.

Proof. Consider neighboring databases $X' = X \cup \{x'\}$ where $|X'| = n' = n + 1$ and $|X| = n$. Let $n'(\cdot,\cdot)$ denote an interval count using X', and let $n(\cdot,\cdot)$ denote an interval count using X. All data points are clipped to $[a, b]$, $o_0 = a$, and $o_{m+1} = b + 1$, so there exists some $[o_{j^*-1}, o_{j^*})$ containing x'. o is nondecreasing and these intervals are half-open, so these intervals do not intersect. Thus, there is exactly one $[o_{j^*-1}, o_{j^*})$ containing x'. Then for $j \neq j^*$, $n'(o_{j-1}, o_j) = n(o_{j-1}, o_j)$ and $(q_j - q_{j-1})n' - (q_j - q_{j-1})n = q_j - q_{j-1}$. Thus

$$u_Q(X, o) = -|n(o_{j^*-1}, o_{j^*}) - n_{j^*}| - \sum_{j\neq j^*} |n(o_{j-1}, o_j) - n_j|$$

and

$$u_Q(X', o) = -|n(o_{j^*-1}, o_{j^*}) + 1 - (q^*_j - q_{j^*-1})(n + 1)| - \sum_{j\neq j^*} |n(o_{j-1}, o_j) - (q_j - q_{j-1})(n + 1)|.$$

The distance between $u_Q(X, o)$ and $u_Q(X', o)$ contributed by the first term is

$$||n(o_{j^*-1}, o_{j^*}) - (q_{j^*} - q_{j^*-1})n| - |n(o_{j^*-1}, o_{j^*}) + 1 - (q_{j^*} - q_{j^*-1})(n + 1)|| = 1 - (q_{j^*} - q_{j^*-1}),$$

and the distance contributed by the second term is

$$\sum_{j\neq j^*} ||n(o_{j-1}, o_j) - (q_j - q_{j-1})n| - |n(o_{j-1}, o_j) - (q_j - q_{j-1})(n + 1)|| \leq \sum_{j\neq j^*} (q_j - q_{j-1}).$$

Thus

$$\begin{aligned} |u_Q(X, o) - u_Q(X', o)| &\leq 1 - (q_{j^*} - q_{j^*-1}) + \sum_{j\neq j^*} (q_j - q_{j-1}) \\ &= 2[1 - (q_{j^*} - q_{j^*-1})]. \end{aligned}$$

The last equality follows from the fact that the sum over all quantile gaps is 1, so the sum over all but the $q_{j^*} - q_{j^*-1}$ gap is $1 - (q_{j^*} - q_{j^*-1})$. The quantity $2[1 - (q_{j^*} - q_{j^*-1})]$ is maximized by minimizing $(q_{j^*} - q_{j^*-1})$, which gives the final sensitivity bound. □

Next, we verify that the finite sampling improvement from Section 3.2 still samples from the correct distribution.

Lemma 5. *$M_{Q'}$ has the same output distribution as M_Q.*

Proof. Recall that the output space for M_Q was $O_\nearrow = \{o = (o_1, \ldots, o_m) \mid a \leq o_1 \leq \cdots \leq o_m \leq b\}$. Define function h on $[a, b]$ by $h(y) = |\{x \in X \mid x < y\}|$. Then $n(u, v) = h(v) - h(u)$, $h(o_0) = h(a) = 0 = i_0$, and $h(o_{m+1}) = h(b + 1) = n = i_{m+1}$. Thus

$$\begin{aligned} u_Q(X, o) &= -\sum_{j\in[m+1]} |n(o_{j-1}, o_j) - n_j| \\ &= -\sum_{j\in[m+1]} |h(o_j) - h(o_{j-1}) - n_j| \\ &= u_{Q'}(X, (h(o_1), \ldots, h(o_m))). \end{aligned}$$

Therefore the normalization term for the distribution defined in Eq. 1 is

$$\begin{aligned} Z_Q &= \int_{O_\nearrow} \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_Q(X, o)\right) do \\ &= \int_{O_\nearrow} \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_{Q'}(X, (h(o_1), \ldots, h(o_m))\right) do. \end{aligned} \tag{2}$$

Note that each $o = (o_1, \ldots, o_m) \in O_\nearrow$ has $o_1 \in [x_{i_1}, x_{i_1+1}), \ldots, o_m \in [x_{i_m}, x_{i_m+1})$ for exactly one $s = (i_1, \ldots, i_m) \in S_\nearrow$. Shorthand this by $o \in s$, and let $g_O(s) = \{o \in O_\nearrow \mid o \in s\}$. Then

$$\begin{aligned}
(2) &= \sum_{s \in S_\nearrow} \int_{g_O(s)} \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_{Q'}(X, (h(o_1), \ldots, h(o_m))\right) do \\
&= \sum_{s \in S_\nearrow} \int_{g_O(s)} \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_{Q'}(X, s)\right) do \\
&= \sum_{s \in S_\nearrow} \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_{Q'}(X, s)\right) \cdot \int_{g_O(s)} do. \qquad (3)
\end{aligned}$$

We focus on the $\int_{g_O(s)} do$ term. If $h(o_1), \ldots, h(o_m)$ are all distinct, i.e. $o_1, \ldots, o_m$ come from distinct intervals between data points, then

$$\int_{g_O(s)} do = \prod_{j=1}^{m} (x_{i_j+1} - x_{i_j}).$$

The remaining (and more complex) case is when $h(o_1), \ldots, h(o_m)$ are not distinct. Suppose $h(o_1), \ldots, h(o_k)$ are not distinct but the remaining $h(o_{k+1}), \ldots, h(o_m)$ are distinct and different from $h(o_1)$. Note that the non-distinct elements are consecutive since $o_1 \leq \cdots \leq o_m$. Then there is some $i \in I$ such that $o_1, \ldots, o_k \in [x_i, x_{i+1})$. Thus the set of valid $o_1, \ldots, o_k$ is exactly $\{(o_1, \ldots, o_k) \mid x_i \leq o_1 \leq \cdots \leq o_k < x_{i+1}\}$.

We need to determine the volume of this set. First, note that the collection consisting of *all* sets of k values from interval i has volume $(x_{i+1} - x_i)^k$. Then, note that the probability that k values selected at random from an interval will be perfectly sorted is $1/k!$; this is the volume of the standard k-simplex, which is the set $\{(x_1, \ldots, x_k) \mid 0 \leq x_1 \leq \cdots \leq x_k \leq 1\}$. Hence, for the set that we are interested in, $\{(o_1, \ldots, o_k) \mid x_i \leq o_1 \leq \cdots \leq o_k < x_{i+1}\}$, the volume is $\frac{(x_{i+1} - x_i)^k}{k!}$.

More generally, this leads us to define the scaling factor γ in Section 3.2:

$$\gamma(s) = \prod_{i \in I} \mathrm{count}_s(i)!$$

where $\mathrm{count}_s(i)$ is the number of times i appears in s, and we take $0! = 1$. γ thus repeats the above scaling process for each interval according to its number of repetitions in $h(o_1), \ldots, h(o_m)$. It follows that for any $s \in S_\nearrow$,

$$\int_{g_O(s)} do = \frac{\prod_{j=1}^{m} (x_{i_j+1} - x_{i_j})}{\gamma(s)}.$$

Returning to our original chain of equalities, we get

$$\begin{aligned}
(3) &= \sum_{s \in S_\nearrow} \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_{Q'}(X, s)\right) \cdot \frac{\prod_{j=1}^{m} (x_{i_j+1} - x_{i_j})}{\gamma(s)} \\
&= Z_{Q'}.
\end{aligned}$$

Turning to the output density f_Q for M_Q, by above

$$f_Q(o) = \frac{1}{Z_{Q'}} \cdot \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_Q(X, o)\right).$$

For any $s \in S_\nearrow$ and any $o, o' \in g_O(s)$ we have $f_Q(o) = f_Q(o')$ and

$$\begin{aligned}
\mathbb{P}_{M_Q}\left[M_Q(X) \in g_O(s)\right] &= \frac{1}{Z_{Q'}} \cdot \int_{g_O(s)} \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_{Q'}(X, (h(o_1), \ldots, h(o_m)))\right) do \\
&= \frac{1}{Z_{Q'}} \cdot \exp\left(\frac{\varepsilon}{2\Delta_{u_Q}} \cdot u_{Q'}(X, s)\right) \cdot \frac{\prod_{j=1}^{m} (x_{i_j+1} - x_{i_j})}{\gamma(s)} \\
&= \mathbb{P}_{M_{Q'}}\left[M_{Q'}(X) \in g_O(s)\right].
\end{aligned}$$

u_Q is constant over $o \in g_O(s)$ for any s, so conditioned on selecting a given $s = (i_1, \ldots, i_m) \in S_\nearrow$, M_Q has a uniform output distribution over increasing sequences from $g_O(s)$, i.e.

$$\begin{aligned} f_Q(o) &= f_Q(o \mid o \in g_O(s)) \cdot \mathbb{P}_{M_Q}\left[M_Q(X) \in g_O(s)\right] \\ &= \prod_{j \in [m]} \frac{\gamma(s)}{x_{i_j+1} - x_{i_j}} \cdot \mathbb{P}_{M_Q}\left[M_Q(X) \in g_O(s)\right] \\ &= \prod_{j \in [m]} \frac{\gamma(s)}{x_{i_j+1} - x_{i_j}} \cdot \mathbb{P}_{M_{Q'}}\left[M_{Q'}(X) \in g_O(s)\right] \\ &= f_{Q'}(o) \end{aligned}$$

where the second and fourth equalities use $\int_{g_O(s)} do = \frac{\prod_{j=1}^m (x_{i_j+1} - x_{i_j})}{\gamma(s)}$. Thus M_Q and $M_{Q'}$ have identical output distributions. □

We now repeat this process for the efficient sampling improvement from Section 3.3.

Theorem 1. *JointExp satisfies ε-differential privacy, takes time $O(mn\log(n) + m^2 n)$, and uses space $O(m^2 n)$.*

Proof. We first verify that *JointExp* samples from $M_{Q'}$, which implies differential privacy. Since the uniform sampling step is unchanged, it suffices to show that the distribution over sampled sequences of intervals is correct.

Let $S_\nearrow(j, i, k) = \{(i_1, \ldots, i_j) \mid i_1 \le \cdots \le i_{j-k} < i_{j-k+1} = \cdots = i_j = i\}$ denote the set of nondecreasing sequences of length j where exactly the last k intervals are equal to i. We will first show that, for all $j \in [m]$, all $i \in I$, and all $k \in [j]$,

$$\alpha(j, i, k) = \sum_{\substack{s=(i_1,\ldots,i_j) \\ s \in S_\nearrow(j,i,k)}} \frac{1}{\gamma(s)} \prod_{j' \in [j]} \phi(i_{j'-1}, i_{j'}, j')\tau(i_{j'}). \tag{4}$$

The base case of $j = 1$ holds by definition, since we let $\alpha(1, i, 1) = \phi(0, i, 1)\tau(1)$ and $\gamma(s) = 1$ when s is a sequence of length one. Before the induction step, we make our optimization for computing α explicit. First, we define

$$\hat{\alpha}(j-1, \cdot) = \sum_{k<j} \alpha(j-1, \cdot, k),$$

a vector of length $n+1$ that can be computed in time $O(mn)$. Then

$$\alpha(j, i, 1) = \tau(i) \sum_{i'<i} \phi(i', i, j) \sum_{k<j} \alpha(j-1, i', k) = \tau(i) \sum_{i'<i} \phi(i', i, j)\hat{\alpha}(j-1, i').$$

Each $\alpha(j, i, 1)$ sums $O(n)$ terms, so a straightforward computation of $\alpha(j, \cdot, 1)$ in its entirety takes time $O(n^2)$. However, we can improve on this by noticing that, after fixing j, each $\phi(i, i', j)$ depends only on $i' - i$. $\phi(\cdot, \cdot, j)$ is therefore a Toeplitz matrix, i.e. a matrix with constant diagonals:

$$\phi(\cdot, \cdot, j) = \begin{bmatrix} \exp\left(-\frac{\varepsilon n_j}{2\Delta_{u_Q}}\right) & \exp\left(-\frac{\varepsilon|1-n_j|}{2\Delta_{u_Q}}\right) & \exp\left(-\frac{\varepsilon|2-n_j|}{2\Delta_{u_Q}}\right) & \cdots & \exp\left(-\frac{\varepsilon|n-n_j|}{2\Delta_{u_Q}}\right) \\ 0 & \exp\left(-\frac{\varepsilon n_j}{2\Delta_{u_Q}}\right) & \exp\left(-\frac{\varepsilon|1-n_j|}{2\Delta_{u_Q}}\right) & \cdots & \exp\left(-\frac{\varepsilon|n-1-n_j|}{2\Delta_{u_Q}}\right) \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & \exp\left(-\frac{\varepsilon|1-n_j|}{2\Delta_{u_Q}}\right) \\ 0 & 0 & 0 & \cdots & \exp\left(-\frac{\varepsilon n_j}{2\Delta_{u_Q}}\right) \end{bmatrix}. \tag{5}$$

It follows that we can use the Fast Fourier Transform (FFT) to multiply $\phi(\cdot, \cdot, j)$ by a vector of length $n+1$ in time $O(n\log(n))$ instead of the typical $O(n^2)$ (a brief reference for this fact appears in the following lecture notes (Bindel, 2019)). Letting $\times$ denote element-wise product, we now rewrite

$$\alpha(j, \cdot, 1) = \tau(\cdot) \times \left(\phi(\cdot, \cdot, j)^T \hat{\alpha}(j-1, \cdot)\right)$$

and use the FFT to compute the second term in time $O(n\log(n))$, since $\phi(\cdot,\cdot,j)^T$ is also Toeplitz. It therefore takes overall time $O(mn\log(n)+m^2n)$ to repeat this for each j and compute $\alpha(\cdot,\cdot,1)$.

Returning to the inductive step, suppose Eq. 4 holds for $j'<j$. Then

$$\begin{aligned}
\alpha(j,i,1) &= \tau(i)(\phi(\cdot,\cdot,j)^T\hat{\alpha}(j-1,\cdot))_i \\
&= \sum_{i'<i}\tau(i)\phi(i',i,j)\hat{\alpha}(j-1,i') \\
&= \sum_{i'<i}\tau(i)\phi(i',i,j)\sum_{k<j}\alpha(j-1,i',k) \\
&= \sum_{i'<i}\tau(i)\phi(i',i,j)\sum_{k<j}\sum_{\substack{s'=(i_1,\ldots,i_{j-1})\\ s'\in S_{\nearrow}(j-1,i',k)}}\frac{1}{\gamma(s')}\prod_{j'\in[j-1]}\phi(i_{j'-1},i_{j'},j')\tau(i_{j'}) \\
&= \sum_{i'<i}\sum_{k<j}\sum_{\substack{s'=(i_1,\ldots,i_{j-1})\\ s'\in S_{\nearrow}(j-1,i',k)}}\tau(i)\phi(i',i,j)\frac{1}{\gamma(s')}\prod_{j'\in[j-1]}\phi(i_{j'-1},i_{j'},j')\tau(i_{j'}) \\
&= \sum_{\substack{s=(i_1,\ldots,i_j)\\ s\in S_{\nearrow}(j,i,1)}}\frac{1}{\gamma(s)}\prod_{j'\in[j]}\phi(i_{j'-1},i_{j'},j')\tau(i_{j'}),
\end{aligned}$$

since every sequence in $S_{\nearrow}(j,i,1)$ consists of a sequence in $S_{\nearrow}(j-1,i',k)$ for some $i'<i$ and $k<j$ with an i appended to the end. Note that the appending of only a single i means that $\gamma(s)=\gamma(s')$. Similarly,

$$\begin{aligned}
\alpha(j,i,k>1) &= \tau(i)\cdot\phi(i,i,j)\cdot\alpha(j-1,i,k-1)/k \\
&= \frac{\tau(i)}{k}\phi(i,i,j)\sum_{\substack{s'=(i_1,\ldots,i_{j-1})\\ s'\in S_{\nearrow}(j-1,i,k-1)}}\frac{1}{\gamma(s')}\prod_{j'\in[j-1]}\phi(i_{j'-1},i_{j'},j')\tau(i_{j'}) \\
&= \sum_{\substack{s=(i_1,\ldots,i_j)\\ s\in S_{\nearrow}(j,i,k)}}\frac{1}{\gamma(s)}\prod_{j'\in[j]}\phi(i_{j'-1},i_{j'},j')\tau(i_{j'}),
\end{aligned}$$

since every sequence in $S_{\nearrow}(j,i,k>1)$ consists of a sequence in $S_{\nearrow}(j-1,i,k-1)$ with an i appended to the end and, when i appears $k-1$ times in sequence s' and s is equal to s' with an i appended to the end, $\gamma(s)=k\gamma(s')$. Thus Eq. 4 holds, and we have the "forward" step: $\alpha(j,i,k)$ is the (unnormalized) mass of nondecreasing length-j sequences ending in k repetitions of i.

Now consider the backward sampling process in *JointExp*. In the first step, we sample a pair

$$\begin{aligned}
(i,k) &\propto \alpha(m,i,k)\phi(i,n,m+1) \\
&= \left[\sum_{\substack{s=(i_1,\ldots,i_m)\\ s\in S_{\nearrow}(m,i,k)}}\frac{1}{\gamma(s)}\prod_{j'\in[m]}\phi(i_{j'-1},i_{j'},j')\tau(i_{j'})\right]\phi(i,n,m+1) \\
&= \sum_{\substack{s=(i_1,\ldots,i_m)\\ s\in S_{\nearrow}(m,i,k)}}\frac{1}{\gamma(s)}\prod_{j'\in[m+1]}\phi(i_{j'-1},i_{j'},j')\prod_{j'\in[m]}\tau(i_{j'}) \\
&\propto \sum_{s\in S_{\nearrow}(m,i,k)}\mathbb{P}_{M_{Q'}}[s]\ ,
\end{aligned}$$

where the second equality uses the fact that we fix $i_{m+1}=n$. Since $\{S_{\nearrow}(m,i,k)\}_{i,k}$ is a partition of $S_{\nearrow}$ (that is, every sequence in $S_{\nearrow}$ appears in $S_{\nearrow}(m,i,k)$ for exactly one value of the pair (i,k)), we conclude that (i,k) is sampled according to the marginal probability that a sequence sampled from $\mathbb{P}_{M_{Q'}}$ ends in exactly k copies of i.

Continuing the backward recursion, if the values of $s_{\text{suffix}} = (i_{j+1}, \ldots, i_m)$ have already been sampled, then at the next step we sample a pair

$$\begin{aligned}
(i < i_{j+1}, k) &\propto \alpha(j, i, k)\phi(i, i_{j+1}, j+1)\tau(i_{j+1}) \\
&= \left[\sum_{\substack{s_{\text{prefix}} = (i_1, \ldots, i_j) \\ s_{\text{prefix}} \in S_{\nearrow}(j,i,k)}} \frac{1}{\gamma(s_{\text{prefix}})} \prod_{j' \in [j]} \phi(i_{j'-1}, i_{j'}, j')\tau(i_{j'}) \right] \phi(i, i_{j+1}, j+1)\tau(i_{j+1}) \\
&\propto \left[\sum_{\substack{s_{\text{prefix}} = (i_1, \ldots, i_j) \\ s_{\text{prefix}} \in S_{\nearrow}(j,i,k)}} \frac{1}{\gamma(s_{\text{prefix}})} \prod_{j' \in [j]} \phi(i_{j'-1}, i_{j'}, j')\tau(i_{j'}) \right] \phi(i, i_{j+1}, j+1)\tau(i_{j+1}) \\
&\cdot \left[\frac{1}{\gamma(s_{\text{suffix}})} \prod_{j'=j+2}^{m+1} \phi(i_{j'-1}, i_{j'}, j') \prod_{j'=j+2}^{m} \tau(i_{j'}) \right] \\
&= \sum_{\substack{s_{\text{prefix}} = (i_1, \ldots, i_j) \\ s_{\text{prefix}} \in S_{\nearrow}(j,i,k)}} \frac{1}{\gamma(s_{\text{prefix}})\gamma(s_{\text{suffix}})} \prod_{j' \in [m+1]} \phi(i_{j'-1}, i_{j'}, j') \prod_{j' \in [m]} \tau(i_{j'}) \\
&\propto \sum_{\substack{s_{\text{prefix}} = (i_1, \ldots, i_j) \\ s_{\text{prefix}} \in S_{\nearrow}(j,i,k)}} \mathbb{P}_{M_{Q'}}\left[s_{\text{prefix}} + s_{\text{suffix}} \right],
\end{aligned}$$

where $+$ denotes sequence concatenation, and we use the fact that, since s_{prefix} and s_{suffix} are nondecreasing and $i_j < i_{j+1}$, $\gamma(s_{\text{prefix}} + s_{\text{suffix}}) = \gamma(s_{\text{prefix}})\gamma(s_{\text{suffix}})$. Again, the set of nondecreasing sequences of length j can be partitioned into disjoint subsets $\{S_{\nearrow}(j, i, k)\}_{i,k}$, thus the pair (i, k) is sampled according to the marginal probability that a sequence sampled from $\mathbb{P}_{M_{Q'}}$, conditional on having the suffix s_{suffix}, has a j-length prefix ending in exactly k copies of i.

Inductively, then, *JointExp* samples a sequence s according to $\mathbb{P}_{M_{Q'}}$. It remains to show that Algorithm 1 satisfies the claimed time and space guarantees.

Time analysis. The first for-loop in Algorithm 1 computes $\alpha(1, \cdot, 1)$ in total time $O(n)$. Each iteration of the second for-loop, over $j = 2, \ldots, m$, computes $\hat{\alpha}(j-1, \cdot)$ in time $O(mn)$, computes $\alpha(j, \cdot, 1)$ in time $O(n \log(n))$ using FFT multiplication, and finally spends $O(m)$ time setting $\alpha(j, i, \cdot)$. The second for-loop thus takes total time $O(m^2 n + mn \log(n))$. Having computed α, each sampling of (i, k) takes time $O(mn)$, so the final sampling takes time $O(m^2 n)$. Summing up, the total time is $O(mn \log(n) + m^2 n)$.

Space analysis. $\hat{\alpha}$ takes $O(mn)$ space, the FFT relying on the Toeplitz expression of $\phi(\cdot, \cdot, j)$ takes space $O(n)$, and α takes $O(m^2 n)$ space. All other variables in the algorithm occupy a constant amount of space, so the overall space usage is $O(m^2 n)$. □

B. Logarithm Trick

In this section, we give details for a more numerically stable logarithmic version of *JointExp*. Recall that we defined $\alpha(1, i, 1) = \phi(0, i, 1)\tau(1)$ and $\alpha(1, i, k) = 0$ for $k > 1$. The former becomes $\ln(\alpha(1, i, 1)) = \ln(\phi(0, i, 1)) + \ln(\tau(1))$ and the latter $\ln(\alpha(1, i, k)) = -\infty$, e.g. using `-numpy.inf` in Python.

We now turn to $\ln(\alpha(j, \cdot, \cdot))$ for $j = 2, \ldots, m$. To set

$$\ln(\hat{\alpha}(j-1, i)) = \ln\left(\sum_{k<j} \alpha(j-1, i, k) \right)$$

$\ln(\alpha)$ terms that have already been computed, we use the following method for summing a vector of quantities a given its component-wise logarithmic form $\ln(a)$

1. Compute the maximum element in the vector: $M_a = \max(\ln(a))$.
2. Component-wise subtract off the maximum element and exponentiate: $a = \exp(\ln(a) - M_a)$.
3. Sum outside of logspace, then return to logspace: $c = \ln(\text{sum}(a))$.
4. Add back the maximum: return $c + M_a$.

An example implementation is `scipy.special.logsumexp` (scipy, 2020).

In the computation of different $\ln(\alpha(j, i, 1))$ using

$$\alpha(j, \cdot, 1) = \tau(\cdot) \times \left(\phi(\cdot, \cdot, j)^T \hat{\alpha}(j-1, \cdot)\right)$$

we want to multiply a Toeplitz matrix A and vector B given their component-wise logarithmic forms $\ln(A)$ and $\ln(B)$ by a similar process. Since A is Toeplitz, we only need to work with its first column $\ln(A_c)$ and first row $\ln(A_r)$. Then we:

1. Compute the maximum element in A_c and A_r and the maximum element in $\ln(B)$: $M_A = \max(\max(\ln(A_c)), \max(\ln(A_r)))$ and $M_B = \max(\ln(B))$.
2. Component-wise subtract off the maximum element and exponentiate: $A_c = \exp(\ln(A_c) - M_A)$, $A_r = \exp(\ln(A_r) - M_A)$ and $B = \exp(\ln(B) - M_B)$.
3. Do the FFT matrix-vector multiplication outside of logspace, then return to logspace: $C = \ln(A \times B)$.
4. Add back the maxima: return $C + M_A + M_B$.

An example implementation for non-FFT matrix multiplication can be found on StackOverflow (StackOverflow, 2014).

C. Sampling by "Racing" Method

The "racing" method is originally due to Ilya Mironov. To the best of our knowledge, full exposition and proofs first appeared in the work of Medina & Gillenwater (2020). We recap their exposition here. The main tool is the following result:

Lemma 11 (Proposition 5 (Medina & Gillenwater, 2020)). *Let $U_1, \ldots, U_N \sim U(0, 1)$ be uniform random samples from $[0, 1]$ and define random variable $R = \arg\min_{[N]} [\ln(\ln(1/U_k)) - \ln(p_k)]$. Then $\mathbb{P}_R[k] = \frac{p_k}{\sum_{j=1}^N p_j}$.*

Lemma 11 enables us to sample from distributions that depend on small probabilities p_k by instead using their logarithms. In combination with the logarithm trick from Appendix B, we avoid dealing with exponentiated terms entirely.

D. Discussion of Other Quantile Algorithms

In this section, we discuss the private quantile estimation algorithms of Dwork & Lei (2009) and Tzamos et al. (2020) and explain why we do not include them in our experiments. Both of these are single quantile algorithms and would require m compositions in order to estimate m quantiles.

Dwork & Lei (2009) define a "propose-test-release" algorithm. Briefly, it discretizes the space into bins of equal width, then computes how many points in the dataset must change to move the true quantile out of its current bin. If this number is too small (specifically, if it is no larger than $\ln^2(n) + 2$, the "test"), then the algorithm does not produce an answer. Otherwise, the answer is the true quantile plus Laplace noise whose scale is six times the bin width ("release").

We can ballpark the accuracy of this method on the uniform data distribution used in our experiments, i.e. $n = 1000$ samples from $U(-5, 5)$. Then $\ln^2(n) + 2 \approx 50$. If we choose a bin width such that the bin with the true median contains 100 points, then it takes at most 50 swaps to move the median out of that bin. We must therefore choose, at a minimum, a bin size such that the bin containing the median contains at least 100 points. Even if we successfully make this choice, then the resulting output will *still* be far less accurate than that of all the other methods tested in the experiments. This is because a successful choice requires a bin width ≥ 1, so the algorithm releases the true median value of ≈ 0 plus Laplace noise with scale 6.

With that scaling, the estimated median is at one of the limits of the $[-5, 5]$ range with probability ≈ 0.434. This means that the estimated median misclassifies roughly 500 out of the 1000 points over 40% of the time, making its expected error in excess of 200 points. For comparison, the algorithms that we test only require lower and upper bounds on the data (not knowledge of the distribution sufficient to choose a good bin width), always output an estimate, and produce average error ≤ 25 for median estimation on uniform data.

We now turn to the private quantile estimation algorithm given by Tzamos et al. (2020). This algorithm is also based on adding (a variant of) Laplace noise to the true median. The first drawback of this method is that its time complexity is $O(n^4)$ (see the footnote accompanying their definition of "TypicalHamming"). This makes it impractical for datasets with more than a few hundred datapoints. The second drawback is the need to select several hyperparameters (R, r, L, C) to determine the specific Laplace noise distribution. While this hyperparameter selection does not affect the privacy guarantee, it does affect the utility. Their utility guarantees assume that the algorithm operator knows these distributional parameters a priori, but this assumption may be hard to satisfy in practice. In contrast, *JointExp* and *AppIndExp* only require the user to provide endpoints.

E. Details For Comparison Algorithms

***AppIndExp*:** Privacy parameters for the m invocations of the exponential mechanism come from the composition guarantee given by Dong et al. (2020). For simplicity, we give a less general (but not weaker) version of their result.

Lemma 12 (Theorem 3 (Dong et al., 2020))**.** *Let mechanism $\mathcal{A}$ consist of m nonadaptive ε-DP applications of the exponential mechanism. Define*

$$t_\ell^* = \left[\frac{\varepsilon_g + (\ell+1)\varepsilon}{m+1}\right]_0^\varepsilon \text{ and } p_{t_\ell^*} = \frac{e^{-t_\ell^*} - e^{-\varepsilon}}{1 - e^{-\varepsilon}}$$

where $[x]_0^\varepsilon$ denotes the value of x clipped to interval $[0, \varepsilon]$. Then $\mathcal{A}$ is (ε_g, δ)-DP for

$$\delta = \max_{0 \leq l \leq m} \sum_{i=0}^{m} \left[\binom{m}{i} p_{t_\ell^*}^{m-i} \cdot (1 - p_{t_\ell^*})^i \cdot \max\left(e^{mt_\ell^* - i\varepsilon} - e^{\varepsilon_g}, 0\right)\right].$$

To apply Lemma 12 with a fixed δ, we use it to compute the largest ε, at a granularity of 0.01, that achieves (ε_g, δ)-DP with some $\delta \leq 10^{-6}$, and we use this value for our experiments. As this is independent of the actual mechanism in question, the time required for this computation is not included in the runtime values reported for *AppIndExp*.

***CSmooth*:** We start with the precise statement of the t-smooth sensitivity of computing a quantile:

Lemma 13 (Proposition 3.4 (Nissim et al., 2007))**.** *Let a and b be client-provided left and right data endpoints. Let X be a database of values $x_1 \leq \ldots \leq x_n$ in $[a, b]$, and for notational convenience define $x_i = a$ for $i < 1$ and $x_i = b$ for $i > n$. Let x_{j^*} be the true value for quantile q on X. Then the t-smooth sensitivity of computing q on X is*

$$\mathcal{S}_q^t(X) = \max_{m=0,\ldots,n} \left(e^{-tm} \cdot \max_{k=0,\ldots,m+1} (x_{j^*+k} - x_{j^*+k-m-1}) \right).$$

Looking at the two $\max$ operations, we can compute $\mathcal{S}_q^t(X)$ in time $O(n^2)$. Nissim et al. (2007) also provide a slightly more involved method for computing $\mathcal{S}_q^t(X)$ in time $O(n \log(n))$. We omit its details here but note that our implementation uses this $O(n \log(n))$ speedup for the fairest time comparison. Next, we specify the exact noise distribution used to generate additive noise in *CSmooth*:

Lemma 14 (Proposition 3 (Bun & Steinke, 2019))**.** *Define the Laplace Log-Normal distribution with shape parameter $\sigma > 0$, $\mathsf{LLN}(\sigma)$, as the distribution for the random variable $Z = X \cdot e^{\sigma Y}$ where $X \sim \mathsf{Lap}(1)$ and $Y \sim N(0, 1)$. Let f be a real-valued function and let $s, t > 0$. Then releasing*

$$f(X) + \frac{\mathcal{S}_f^t(X) \cdot Z}{s}$$

satisfies $\frac{\varepsilon^2}{2}$-CDP for $\varepsilon = \frac{t}{\sigma} + e^{1.5\sigma^2} s$.

Once we fix the desired CDP privacy parameter $\frac{\varepsilon^2}{2}$, to apply Lemma 14 we must still select $t, s, \sigma > 0$. We follow the selection method given in Sections 3.1.1 and 7.1 of Bun & Steinke (2019), omitting most of the details. First, for each of a sequence of values for t, we set $s = e^{-1.5\sigma^2}(\varepsilon - t/\sigma)$ and numerically solve for σ as a root of the polynomial $\frac{5\varepsilon}{t}\sigma^3 - 5\sigma^2 - 1 = 0$. Repeating this process for each t provides a collection of (t, s, σ) triples without touching the database X. Given these triples (t, s, σ), we finally select one to minimize variance $\frac{2\mathcal{S}_f^t(X)^2}{e^{-5\sigma^2}(\varepsilon - t/\sigma)^2}$.

We pause to note that this last minimization of variance repeatedly touches X to compute $\mathcal{S}_f^t(X)$ for different t. As this is not differentially private, we executed this non-private selection process once using data drawn from the standard Gaussian $N(0, 1)$ and used the resulting values for *CSmooth* experiments on our datasets. In practice, after starting from a wide range for t of 150 logarithmically spaced values between 10^{-10} and 10, we found that the values selected for t clustered in a narrow subinterval across both data drawn from $N(0, 1)$ and data drawn from our other experiment distributions. We therefore view the distribution-specific selection of t as contributing relatively little to the final error of *CSmooth*.

In more detail, the actual t selection process in our experiments is to use the variance-minimizing selection process described in Section 5.1 for each quantile in sets of quantiles ranging from $m = 1$ to $m = 29$ for $\varepsilon = 1$. The range for t is 50 logarithmically spaced values between 0.01 and 1. Each trial used 1000 samples drawn from $N(0, 1)$ with data lower bound -100 and data upper bound 100. Below, we record the t selected for each quantile and quantile range, averaged across 5 trials. Each color represents a different set of quantiles, and each point for each color represents the t selected for a single quantile.

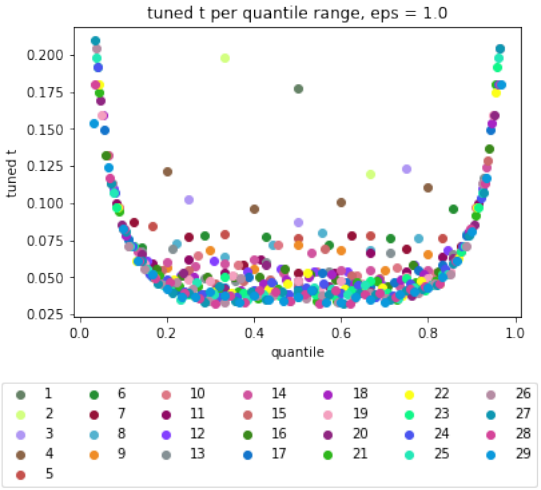

Figure 5. Tuned t used for *CSmooth* across different quantile ranges. For example, we used $t \approx 0.13$ for $m = 3$, and $q = 0.5$ (the magenta dot in the middle of the plot).

***AggTree*:** *AggTree*'s hyperparameters are the tree height h and branching factor b. We tuned these parameters over the range $\{2, 3, \ldots, 15\}$ and $\{1, 2, \ldots, 15\}$ respectively. As with *CSmooth*, we used $N(0, 1)$ data. The following two tables summarizes the values tuned over 50 trials of 1000 data points each.

# quantiles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
height	4	3	3	3	2	3	3	3	3	3	3	3	3	3	3
branching parameter	4	6	6	9	14	10	7	7	10	10	8	7	7	12	10

# quantiles	16	17	18	19	20	21	22	23	24	25	26	27	28	29
height	3	3	3	3	3	3	3	3	3	3	3	3	3	3
branching parameter	10	10	10	7	10	10	7	10	12	12	12	10	10	12

Table 1. Tuned height and branching parameters across number of quantiles.

F. Distance Error Experiments

We conclude with experiments using a distance error metric, which computes the average ℓ_1 distance between the vectors of estimated and true quantiles: given quantile estimates $\hat{o}_1, \ldots, \hat{o}_m$ and true values $o_1, \ldots, o_m$, the error is $\|\hat{o} - o\|_1/m$. In this setting, we re-tune *AggTree*'s hyperparameters using the distance metric, although the results are essentially the same:

# quantiles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
height	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
branching parameter	8	6	6	10	9	9	10	10	12	12	10	10	7	10	8

# quantiles	16	17	18	19	20	21	22	23	24	25	26	27	28	29
height	3	3	3	3	3	3	3	3	3	3	3	3	3	3
branching parameter	8	10	7	10	10	12	8	10	12	10	7	5	7	5

Table 2. Tuned height and branching parameters across number of quantiles (distance error).

The final error plots appear below. Note that the algorithms that rely on the exponential mechanism (*AppIndExp* and *JointExp*) at some point exhibit a sharp increase in error as m grows. This is because these algorithms eventually end up sampling from a distribution that favors the extreme intervals containing the domain endpoints, and – unlike misclassification error – distance error strongly penalizes these outputs. Nonetheless, *JointExp* still achieves the strongest performance for a wide range of m.

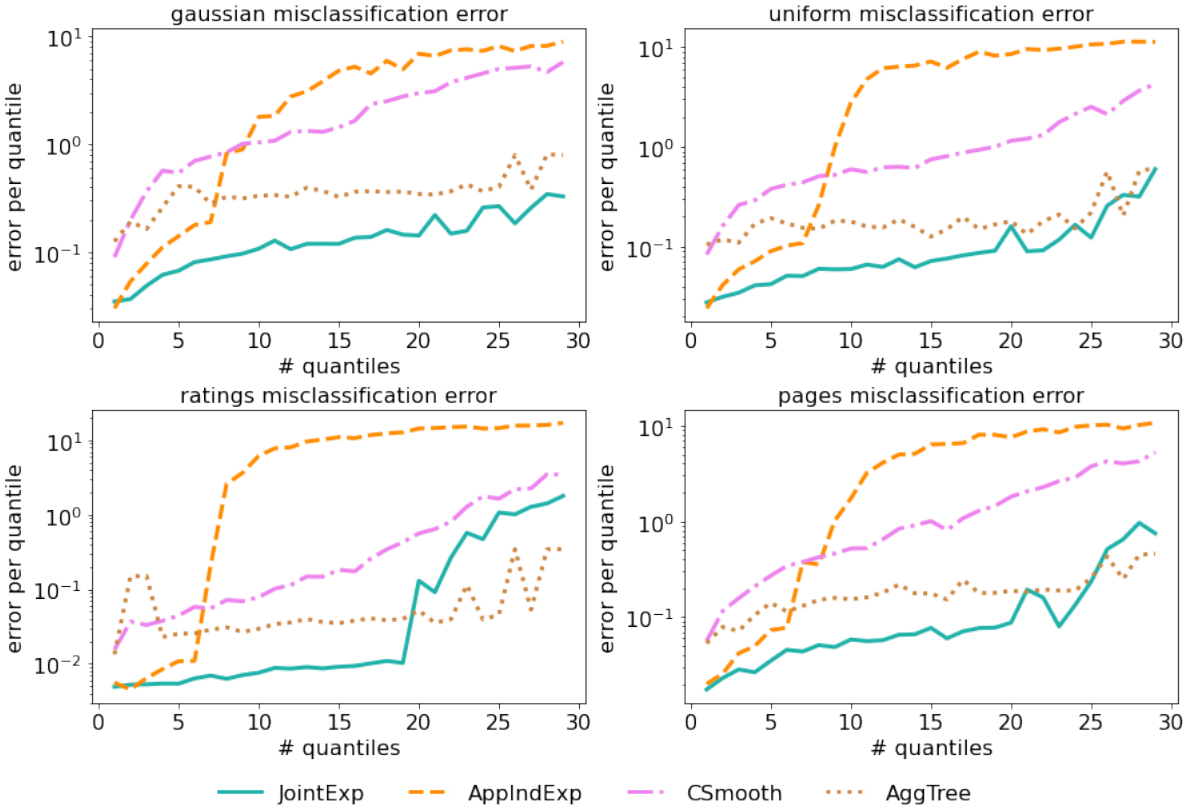

Figure 6. Distance error vs # quantiles m for $\varepsilon = 1$, averaged across 50 trials of 1,000 samples.