Bit-efficient Numerical Aggregation and Stronger Privacy for Trust in Federated Analytics

Graham Cormode, Igor L. Markov
gcormode@fb.com, imarkov@fb.com

Abstract

Private data generated by edge devices — from smart phones to automotive electronics — are highly informative when aggregated but can be damaging when mishandled. A variety of solutions are being explored but have not yet won the public's trust and full backing of mobile platforms. We propose numerical aggregation protocols that empirically improve upon prior art, while providing comparable local differential privacy guarantees. Sharing a single private bit per value supports (i) privacy metering that enables privacy controls and (ii) guarantees that are not covered by differential privacy. We emphasise ease of implementation, compatibility with infrastructure, and compelling empirical performance.

1 Introduction

Smart phones and smart watches, fitness trackers, automotive electronics, building sensors, and other edge devices collect large amounts of private data, including locations, timestamps, behaviors, personal preferences, as well as data related to financial and medical information. To limit the impact and potential damage caused by a private datum, aggregation (e.g., a sample mean, the 90th percentile or a histogram) is used whenever possible and often suffices. A potential danger is that aggregated data may nevertheless reveal some information about individual contributors in rare cases. To prevent that, *federated analytics* methods [10] use aggregation protocols with mathematical guarantees (*differential privacy* [6] for *federated learning* [9, 3, 7]), where the noise is added before aggregation to provide *plausible deniability* through *randomized response* techniques [12].

The mathematical sophistication of existing methods complicates audit and verification, increases the chances of implementation mistakes, and leaves many possible attack vectors. Our work addresses these challenges, while absorbing or extending prior solutions. We do not assume secure hardware but can make use of it; likewise, we do not introduce novel differentially private mechanisms, but provide a DP guarantee by leveraging an off-the-shelf mechanism.

Privacy metering. We propose to meter private data not at the value level (such as an integer representing someone's current longitude), but at the bit level. Rather than transmit an entire private value with noise added, our aggregation protocols only transmit a single private bit (and limit subsequent bits per value and per client).

Bit-efficient numerical aggregation. Our mathematical contributions are the introduction and analysis of more efficient techniques for the computational estimation of means, variances, etc. Prior LDP techniques either operate directly on real numbers, e.g., adding Laplace noise, or produce some noisy discrete values as a result of rounding, range checks, comparisons to thresholds, etc. In contrast, we approximate real numbers with fixed-point (or integer) representations, expand them in binary, select some of the resulting bits, and postprocess those bits before communicating them. We observe that the efficiency of existing approaches often relies on knowing tight bounds on the range in which the values fall. We relax this assumption and allow our approach to adapt to the distribution of values observed in practice. More details are presented in the full version of this work [2].

Preprint. Under review.

Prior work. Several *LDP mechanisms* add noise at the bit level, e.g. Duchi *et al.* [5] combine randomized response [12] with randomized rounding. Similar ideas have been deployed by Microsoft for Windows app usage data collection [4]. Subsequent work by Wang *et al.* modifies the procedure to sample a value so that values closer to the input are chosen with higher probability than those that are further away, referred to as the "piecewise" mechanism [11].

Communication-efficient numerical aggregation (regardless of privacy) was considered for low-bandwidth sensor networks [8] Dealing with multi-dimensional data, distributed ML applications can leverage bit-level efficiency to reduce their communication. Ben-Basat *et al.* analyze leading approaches for estimating a real value using a single bit sent from a client to a server, as a function of the amount of shared randomness between the two parties [1]. In our setting, the relevant point of comparison is given by *subtractive dithering*, wherein each client samples a random threshold and indicates whether its value is above or below it.

The need for adaptive protocols. All these methods assume inputs in the range $[0, 1]$ or, equivalently, in some range $[L, H]$ mapped to $[0, 1]$ via $f(x) = \frac{x-L}{H-L}$. Assuming loose bounds on input values has a negative impact on accuracy: when methods that are optimal for $[0, 1]$ are applied to $[L, H]$, the variance of their estimates scales with $(H - L)^2$ [1]. More promising are protocols that adapt to the data distribution and "zoom in" on the range where the data truly lies. We implement such adaptation in our protocols using few rounds and show (i) sharper analytical bounds for variance, backed by (ii) reduced variance in simulations. When combined with the intuitive nature of the bit-level privacy metering and ease of achieving a formal ϵ-LDP guarantee, bit-pushing becomes an attractive option for mean estimation and related tasks.

2 The Bit-pushing Approach

Basic bit pushing algorithm. Assume that each client i out of N owns a private value x_i: we work with b-bit integer and fixed-point values. In the narrative below, we assume non-negative integers, but this is not a limitation of the approach. Our goal is to estimate the mean $\bar{x} = \sum_{i=1}^{N}(x_i/N)$. We write $x^{(j)}$ to denote the j'th bit in the binary representation of x, and $\bar{x}^{(j)}$ to denote the j'th bit of the mean, $\bar{x}$. In the basic form of bit pushing, each client selects bit j with probability p_j, and sends the value of their input at this bit location, as the pair $\langle x_i^{(j_i)}, j_i \rangle$. Algorithm 1 (in the Appendix) gives the corresponding pseucocode for bit pushing, given a probability vector p to sample bits with.

Lemma 1. *The basic bit pushing protocol provides an estimate that is unbiased and has variance equal to* $\frac{1}{N}\sum_{j=0}^{b-1} 4^j \bar{x}^{(j)}(1 - \bar{x}^{(j)})/p_j$.

Corollary 2. *If each client sends* b_{send} *bits, the variance decreases to* $\frac{1}{Nb_{\text{send}}}\sum_{j=0}^{b-1} \frac{4^j \bar{x}^{(j)}(1-\bar{x}^{(j)})}{p_j}$

The proofs of all claims are presented in the Appendix. This result relies on the fact that the mean is a linear function of the values at each bit location. It seems intuitive that higher-order bits should have greater probability of being sampled, since they contributed more highly to the computation. Some natural choices are $p_j \propto 2^j$, or more generally, $p_j \propto c^j = 2^{\alpha j}$ for some c or α.

Lemma 3. *The variance of the bit-pushing estimator is minimized by picking* $p_j = \sum_{j=0}^{b-1} \frac{\sqrt{\beta_j}}{B}$, *where* $\beta_j = 4^j \bar{x}^{(j)}(1 - \bar{x}^{(j)})$, *and* $B = \sum_{j=0}^{b-1} \sqrt{\beta_j}$.

From Lemma 1 and our independence assumption, we have $\beta_j = \bar{x}^{(j)}(1 - \bar{x}^{(j)})4^j$, where $\bar{x}^{(j)}$ denotes the mean value at the j'th bit index. If we simply bound the contribution of $\bar{x}^{(j)}(1 - \bar{x}^{(j)})$ values by $\frac{1}{4}$, so that $\beta_j = \frac{1}{4}4^j$, this leads us to set $p_j = 2^j/(2^b - 1)$, and so we obtain

$$\mathbb{V}[X] \leq \frac{1}{N}\sum_j (2^b - 1)\bar{x}^{(j)}(1 - \bar{x}^{(j)})2^j < \bar{x}(2^{b-2})/N. \tag{1}$$

If the inputs make use of most input bits, then $\bar{x}$ is reasonably large, i.e., $\bar{x} \propto 2^b$, and so $\mathbb{V}[X] \propto \bar{x}^2/N$. Hence, the expected absolute error will be of magnitude $\bar{x}/\sqrt{N}$. Sending $b_{\text{send}} > 1$ bits per client would further reduce this absolute error by a factor of $1/\sqrt{b_{\text{send}}}$.

Adaptive Bit Pushing. A more sophisticated approach is to use a first round of bit pushing to estimate the bit means $\bar{x}^{(j)}$. That is, we first choose a set of sampling probabilities p_j independent

of the input, and ask a δ fraction of the clients to report an input bit according to this distribution. From these reports, we estimate $\bar{x}^{(j)}$ as $\hat{x}^{(j)}$ for all j, and use these estimates to compute a new set of weights based on $\beta'_j = \hat{x}^{(j)}(1-\hat{x}^{(j)})4^j$. We can then perform a second round of bit pushing using sampling probabilities $p_j = \sqrt{\beta'_j}/\sum_{j=0}^{b-1}\sqrt{\beta'_j}$ for the remaining $1-\delta$ fraction of clients. To instantiate this two-round approach, we need to determine (i) what split parameter δ to apply; and (ii) how to choose the initial weights β_j. Naively, we might choose $\delta = \frac{1}{2}$ to balance accuracy of learned β'_js and accuracy of reported results. For β_js, we might default to choosing $\beta_j = 4^j$ (and hence $p_j \propto 2^j$), according to the above argument. Our analysis guides the choice of $\delta = \frac{1}{3}$. Algorithm 2 in the Appendix provides pseudocode for the adaptive bit pushing approach.

Lemma 4. *The variance of adaptive bit pushing is bounded by* $\frac{b_{\max}\sigma^2}{N} + O(b_{\max}4^{b_{\max}}/N^{3/2})$*, where* $b_{\max}$ *is the index of the highest-order bit that is non-zero in the input.*

Comparison to alternate approaches. The benefit of adaptive bit pushing can be most easily understood when we have only a loose estimate of b, the number of bits to represent the input values. For methods which scale the input down to the range $[0,1]$ and then scale the estimated fraction back up, the variance of the resulting estimate is proportional to $(2^b)^2$. For (non-adaptive) bit pushing, it is proportional to $2^b\bar{x}$, as shown in (1). Since adaptive bit-pushing allows us to identify any bits j with $\bar{x}_j = 0$, we can bound the variance of the estimate by $2^{b_{\max}}\bar{x}$, or use the above analysis to argue that the variance is proportional to $b_{\max}\sigma^2$ plus lower order terms. Compared to the bound (1) for our non-adaptive protocol, variance is reduced by a factor of at least $2^{b-b_{\max}}$.

Local Differential Privacy. It is straightforward to give bit pushing an ϵ-LDP guarantee: we apply randomized response [12] to each bit before it is sent, and unbias the results at the server side. The variance of this unbiased estimator is $\frac{\exp\epsilon}{(\exp\epsilon-1)^2}$. In contrast to the above analysis, this variance is independent of the bit means $\bar{x}^{(j)}$. When we apply randomized response to bit pushing, where we assign p_jN clients to report on bit j, which is scaled by 2^j, then we obtain a total bound on the variance of $\sum_{j=0}^{b-1}\frac{4^j}{p_jN}\frac{\exp\epsilon}{(\exp\epsilon-1)^2}$. This is optimized according to the above argument by choosing $p_j = 2^j/(2^b-1)$, which yields a total variance bound of $O\left(\frac{4^b}{N}\frac{\exp\epsilon}{(\exp\epsilon-1)^2}\right)$. For small $\epsilon < 1$, this expression is $O(4^b/N\epsilon^2)$, and so the expected absolute error is $O(2^b/\epsilon\sqrt{N})$.

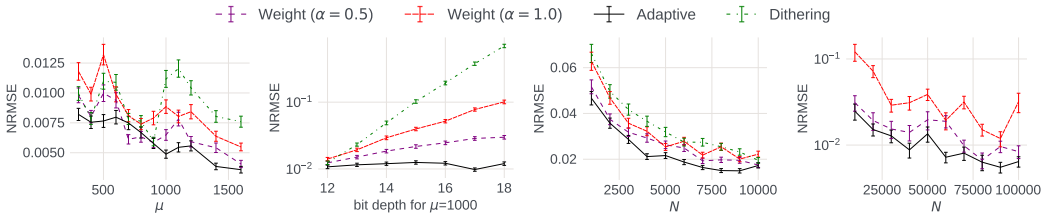
Variance estimation. The (empirical) variance is a fundamental primitive that can be easily expressed as an expectation, and hence computed via bit-pushing. Specifically, we can write two mathematically equivalent expressions for the variance: $\mathbb{V}[X] = \mathbb{E}[(X-\mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$. The former gives better results when using approximate means, so we adopt it in our experiments.

3 Experiments

We simulate the computational costs and accuracy of mean estimation for bit-pushing—the two-round adaptive bit pushing ("adaptive"), and the single round approach based on a fixed allocation of weights to bits ("weighted")—and compare against subtractive dithering [1] ("dithering"). We perform experiments on both human-generated and synthetic data. To generate synthetic data, we draw values from Normal, uniform and exponential distributions with varying parameters, as specified below. The human-generated data (*real data* for short) are the reported ages from publicly-available de-identified data from the US Census[1]. Our main focus is on comparing the accuracy of the different techniques, measured by the normalized root-mean-squared error (NRMSE): in each experiment, we compare the true (empirical) value of the mean μ to the estimate $\hat{x}$, and compute the mean of the squared difference over 100 independent repetitions, then divide by the true mean μ for normalization. When error bars are shown on our plots, they indicate the standard error of these repetitions. Our default number of clients (10K) is representative of typical federated scenarios. Initial parameter tuning experiments lead us to set $\delta = 1/3$, and $\alpha = 0.5$ or $\alpha = 1.0$.

Figure 1a shows the accuracy as we vary the mean μ of the input (Normal) distribution. There is a general trend for the normalized error to decrease as μ increases, since the normalizing constant

[1] `https://archive.ics.uci.edu/ml/datasets/Census-Income+%28KDD%29`



(a) Mean accuracy vs. μ (b) Mean accuracy vs. bits (c) Mean accuracy vs. N (d) Variance accuracy vs.N

Figure 1: Accuracy experiments on Normal data varying μ ($\sigma = 100$) and on real data (varying N)

increases faster than the magnitude of the errors. For the dithering approach, there is a step-up in error around powers of two, the point at which we increase the bound on the input values by a factor of 2. In a single round, choosing weights based on $\alpha = 0.5$ generally leads to more accurate results. Across the whole domain, the adaptive approach reliably achieves the least error. The absolute values of NRMSE are also encouraging: the error is typically around 1-2% of the true mean value.

We study the importance of finding accurate bounds on the magnitude of the quantities involved in Figure 1b. We vary the "bit depth", i.e., the number of bits b used in the bit-pushing algorithms (so 2^b is the bound used for the dithering approach). We see that all the one-round approaches grow in error as b increases: less so for $\alpha = 0.5$, since less weight is apportioned to the (vacuous) high order bits than in the $\alpha = 1.0$ case. The adaptive approach can identify the redundant bits in the first round, and discard them in round two, so it is largely oblivious to the increase in bit depth.

Experiments varying N on real data are shown in Figure 1. The normalized error for both mean (Figure 1c) and variance (Figure 1d) estimation tends to decrease as N increases, broadly consistent with the predicted dependence on $N^{-1/2}$. Variance estimation is the harder task, as evidenced by the substantially larger error values, even though we allocate a larger cohort of 100,000 clients to this task. Here, the dithering approach is orders of magnitude worse, due to its inability to adapt to the scale of the input values. The adaptive approach achieves the best accuracy overall, particularly with a larger user cohort: it achieves normalized errors in the 1-2% range. Experiments for DP noise are shown in Appendix B, and additional results in the full paper [2].

4 Conclusions and Perspectives

Notions of optimality for single-bit estimates have been explored in the recent literature, and methods such as *subtractive dithering* were shown optimal [1]. Hence, it may look surprising that bit pushing empirically outperforms those methods. This apparent paradox stems from the assumptions made in prior proofs of optimality. In particular, optimality is invalidated when the true mean can be narrowed down further within the fixed subrange $[0, 1]$, and this bracketing is performed by adaptive bit pushing using the same type of inputs as other protocols. Limitations of this approach are the flip side of its advantages: when a tight bound on the values *is* known in advance, then bit pushing attains similar accuracy to existing methods . However, accuracy is not the only relevant metric. In many cases, a single bit of the client's input does not reveal any sensitive information, and so bit pushing alone can provide an intuitive privacy promise to non-experts. When some bits of a value can be privacy-revealing (say, disclosing if a value is above or below a threshold), plausible deniability for communicated bits is ensured using differential privacy (randomized response) [12].

Privacy metering at the bit level can be used in conjunction with bit pushing and differential privacy to provide stronger privacy guarantees and help the general public improve trust in technology. Our work shows how to communicate fewer private bits when aggregating data, and metering private bits shared by an edge device may provide a language more accessible and more convincing to the general public than the language of differential privacy. Privacy infrastructure is needed to turn our proposal into a reality. Platforms for federated analytics and federated learning should provide configurable aggregation services that would package private bits into larger network packets and provide layers of separation to rule out the mixing of private and non-public bits. With such infrastructure, it should be relatively easy to add privacy metering and monitoring, potentially giving the users greater control of their data privacy.

References

[1] Ran Ben-Basat, Michael Mitzenmacher, and Shay Vargaftik. How to send a real number using a single bit (and some shared randomness). *CoRR*, abs/2010.02331, 2020.

[2] Graham Cormode and Igor L. Markov. Bit-efficient numerical aggregation and stronger privacy for trust in federated analytics. *CoRR*, abs/2108.01521, 2021.

[3] Differential Privacy Team at Apple. Learning with privacy at scale. `https://machinelearning.apple.com/research/learning-with-privacy-at-scale`, December 2017.

[4] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *NeurIPS*, pages 3571–3580, 2017.

[5] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Minimax optimal procedures for locally private estimation. *Journal of the American Statistical Association*, 113(521):182–201, 2018.

[6] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4), 2014.

[7] Matthias Paulik et al. Federated evaluation and tuning for on-device personalization: System design & applications. `arXiv:2102.08503`, February 2021.

[8] Zhi-Quan Luo. Universal decentralized estimation in a bandwidth constrained sensor network. *IEEE Trans. Inf. Theory*, 51(6):2210–2219, 2005.

[9] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`, April 2017. Google AI Blog.

[10] Daniel Ramage and Stefano Mazzocchi. Federated analytics: Collaborative data science without data collection. `https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html`, May 2020. Google AI Blog.

[11] Ning Wang, Xiaokui Xiao, Yin Yang, Jun Zhao, Siu Cheung Hui, Hyejin Shin, Junbum Shin, and Ge Yu. Collecting and analyzing multidimensional data with local differential privacy. In *IEEE International Conference on Data Engineering*, pages 638–649. IEEE, 2019.

[12] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

Algorithm 1: Basic bit pushing algorithm

Input : No. of bits b, bit weights p, no. of clients N
Output: (Result r, mean of bits m, sum of bits s)

Initialize result $r = 0$
for $j = 0$ **to** $b - 1$ **in parallel do**
  Contact $c[j] = p[j] \cdot N$ clients to request bit j
  Gather weighted sum of bits as $s[j]$
  Compute bit means $m[j] = s[j]/c[j]$
  $r \leftarrow r + (2^j \cdot m[j])$
return (r, m, s)

Algorithm 2: Adaptive bit pushing algorithm

Input : No. of bits b, no. of clients N, parameters α, γ, δ
Output: Result r

`/* Round 1:` `*/`
for $j = 0$ **to** $b - 1$ **do**
  Compute $p_1[j] = (2^j)^\gamma$
Normalize p_1: $p_1 \leftarrow p_1/\text{sum}(p_1)$
Run basic bit pushing to estimate bit means: $(r_1, m_1, s_1) = \texttt{BitPushing}(b, p_1, \delta N)$
`/* Round 2:` `*/`
for $j = 0$ **to** $b - 1$ **do**
  Compute $p_2[j] = (4^j \cdot m_1[j] \cdot (1 - m_1[j])^\alpha$
Normalize p_2: $p_2 \leftarrow p_2/\text{sum}(p_2)$
Run basic bit pushing based on the computed probabilities:
  $(r_2, m_2, s_2) = \texttt{BitPushing}(b, p_2, (1 - \delta)N)$
`/* Final aggregation:` `*/`
Combine means $m_3 = (s_1 + s_2)/(\delta N * p_1 + (1 - \delta)N * p_2)$
for $j = 0$ **to** $b - 1$ **do**
  $r \leftarrow r + 2^j \cdot m_3[j]$
return r

A Omitted Proofs

Proof of Lemma 1. Let $X^{(j)}$ denote the distribution of the j'th bit value. We can assume that each $X^{(j)}$ follows a Bernoulli distribution with parameter $\mathbb{E}[X^{(j)}]$. Assuming the quasi-Monte Carlo case, where bit j is reported on by exactly Np_j clients, our estimate $\hat{X}^{(j)}$ is the mean of these Np_j reports. Clearly $\mathbb{E}[\hat{X}^{(j)}] = \mathbb{E}[X^{(j)}]$, which, by linearity of expectation, is $\bar{x}^{(j)}$. Our estimate X is the sum of these bit means, weighted by 2^j, so $X = \sum_{j=0}^{b-1} 2^j \hat{X}^{(j)}$ (applying linear decomposition), and by definition,

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{j=0}^{b-1} 2^j X^{(j)}\right] = \sum_{j=0}^{b-1} 2^j \bar{x}^{(j)} = \bar{x}. \tag{2}$$

For bit j, each report on this bit is assigned weight 2^j. The corresponding contribution to the variance is $\mathbb{V}[2^j X^{(j)}] = 4^j \bar{x}^{(j)}(1 - \bar{x}^{(j)})$. Averaged over the Np_j reports, the contribution to the variance from the estimate of bit j is $4^j \bar{x}^{(j)}(1 - \bar{x}^{(j)})/(Np_j)$, so the overall variance of the estimator is

$$\mathbb{V}[X] = \sum_{j=0}^{b-1} \frac{4^j}{Np_j} \bar{x}^{(j)}(1 - \bar{x}^{(j)}) := \frac{1}{N} \sum_{j=0}^{b-1} \frac{\alpha_j}{p_j} \tag{3}$$

□

Proof of Corollary 2. This result follows immediately by adapting the previous proof to average over $(Np_j b_{\text{send}})$ samples for bit j. □

Proof of Lemma 3. For a fixed budget of bit samples, we seek to minimize $\mathbb{V}[X] = \frac{1}{N}\sum_j \frac{\beta_j}{p_j}$ with $\beta_j, p_j > 0$. To optimize variance in terms of p_j such that $\sum_j p_j = 1$, we perform unconstrained optimization of

$$f(p_1, \ldots, p_{k-1}) = \frac{\beta_k}{1-\sum_{j=0}^{k-1} p_j} + \sum_{j=0}^{k-1} \frac{\beta_j}{p_j} \tag{4}$$

Looking for a local extremum inside the probability simplex, we obtain

$$\forall i = 1..k-1, \quad \frac{\partial f(p_1,\ldots,p_k)}{\partial p_i} = \frac{\beta_k}{(1-\sum_{j=0}^{k-1} p_j)^2} - \frac{\beta_i}{p_i^2} = 0 \tag{5}$$

Therefore

$$\forall i, l \quad \frac{\beta_i}{p_i^2} = \frac{\beta_l}{p_l^2} \qquad \Rightarrow \qquad p_i/p_l = \sqrt{\beta_i/\beta_l} \tag{6}$$

and this extends to $l = k$ via renumbering. Therefore, $p_j = \sqrt{\beta_j}\frac{p_k}{\sqrt{\beta_k}}$, and to find p_j, we can just L_1-normalize the vector of $\sqrt{\beta_j}$. To confirm that this unique critical point is the global minimum, we compute

$$\forall i, j \quad \frac{\partial \mathbb{V}}{\partial p_i \partial p_j} = \left\{ \begin{array}{ll} 0, & \text{for } i \neq j \\ 2\beta_j/p_j^3 N & \text{for } i = j \end{array} \right\} \tag{7}$$

Given that $p_j, \beta_j > 0\ \forall j$, the Hessian is positive semidefinite. □

Proof of Lemma 4. With adaptive bit pushing, the first round allows us to find accurate estimates for each of the β_j parameters in (3), and we proceed by substituting our choice of p_j into (3). We first assume that the estimates from the first round give exact values for $p_j \propto \sqrt{\beta_j}$. Write $B = \sum_{j=0}^{b-1} \sqrt{\beta_j}$, so that $p_j = \sqrt{\beta_j}/B$. Then (3) sets

$$\mathbb{V}[X] = \frac{1}{(1-\delta)N} \sum_{j=0}^{b-1} B\frac{\beta_j}{\sqrt{\beta_j}} = \frac{B}{(1-\delta)N} \sum_{j=0}^{b-1} \sqrt{\beta_j} = \frac{B^2}{(1-\delta)N} \tag{8}$$

Next, we can observe that, using the Cauchy-Schwarz inequality on a vector of up to $b_{\max}$ different β_j values,

$$B^2 \leq b_{\max} \sum_{j=0}^{b_{\max}-1} \beta_j = b_{\max} \sum_{j=0}^{b_{\max}-1} 4^j \bar{x}^{(j)}(1-\bar{x}(j)) := b_{\max}\sigma^2 \tag{9}$$

where, σ^2 is the variance of the input distribution since, by linearity of expectation, we can decompose $\sigma^2 = \sum_{j=0}^{b-1}(\sigma^2)^{(j)} = \sum_{j=0}^{b-1} 4^j \bar{x}^{(j)}(1-\bar{x}^{(j)})$. Hence, our estimate is as efficient as the trivial (non-private) estimator of taking the mean of N samples, up to the factor of $b_{\max}/(1-\delta)$.

We now consider the effect of sampling δN clients in the first round. For a given bit j, if we estimate this based on a sample of size s, then the error in our estimate of $\bar{x}^{(j)}$ will be proportional to $1/\sqrt{s}$, from standard sampling bounds. Then our estimate of β_j (as $\hat{\beta}_j$) will accordingly have an error of $O(4^j/\sqrt{s})$. Putting this into our expression for σ^2 yields

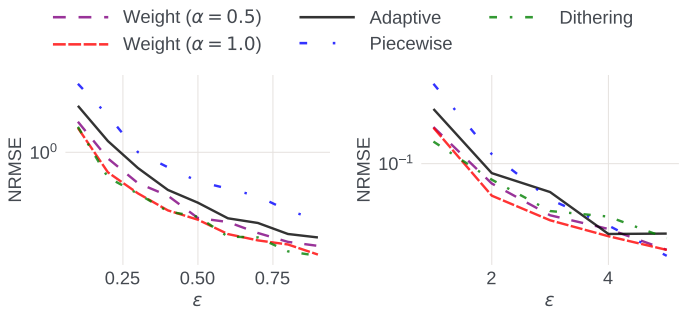
$$\hat{\sigma}^2 = \sum_{j=0}^{b_{\max}} \hat{\beta}_j = \sum_{j=0}^{b_{\max}} \left(\beta_j + O(4^j/\sqrt{s})\right) < \sigma^2 + O\left(4^{b_{\max}}\frac{1}{\sqrt{s}}\right). \tag{10}$$

Hence, the contribution to the total variance from this error term is, from (8), (9), and (10), proportional to $\frac{4^{b_{\max}}}{(1-\delta)N\sqrt{\delta N}}$. This term is minimized, as a function of δ, if we maximize the expression $(1-\delta)\sqrt{\delta}$. Writing $z = \sqrt{\delta}$, we aim to maximize $z - z^3$. We differentiate to obtain $1 - 3z^2 = 0$, and so set $z^2 = \delta = 1/3$.

Consequently, our bound on the variance is

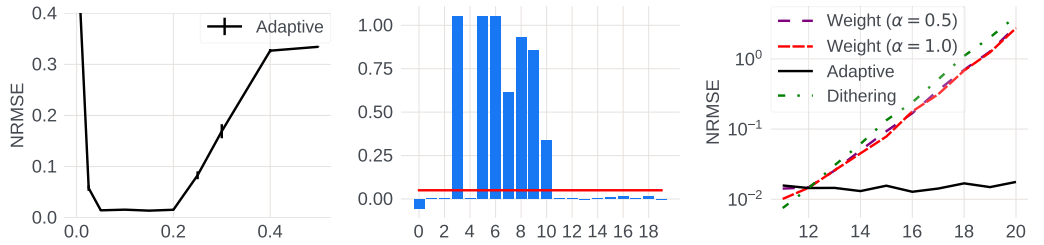
$$b_{\max}\sigma^2/N + O(b_{\max}4^{b_{\max}}/N^{3/2}). \tag{11}$$

Provided the variance in the high-order bits is significant (e.g., if $(\sigma^2)^{(b_{\max})}$ is at least a constant), then this is dominated by the first term of σ^2/N. □



(a) Mean estimation, $\epsilon < 1$ (b) Mean estimation, $\epsilon > 1$

Figure 2: Differential privacy experiments on real data



(a) Varying *bit squashing* threshold (b) Histogram of bit means (c) Varying bit depth for DP $\epsilon = 3.0$

Figure 3: Accuracy experiments on synthetic data with differential privacy

B Experimental Results with Differential Privacy

We study the impact of providing a differential privacy (DP) guarantee on mean estimation. We include the "piecewise" mechanism [11], as well as our previous one-bit methods augmented with randomized response to provide a DP guarantee. Figure 2 shows the NRMSE accuracy on synthetic data as we vary the privacy parameter ϵ, split into two regimes: high privacy ($\epsilon < 1$), Figure 2a, and moderate privacy ($\epsilon \geq 1$, Figure 2b). On a log scale plot, the lines are fairly closely clustered, but we see that in this experiment, the single round approach with $\alpha = 1.0$ achieves the least error. Only when $\epsilon > 5$ do we see points where the piecewise approach achieves lower error. Note that the absolute value of NRMSE for mean estimation is much larger than without DP noise until $\epsilon = 5$. This is all consistent with our theoretical analysis, where we showed that the variance depends only on ϵ (as $\frac{\exp \epsilon}{(\exp \epsilon - 1)^2}$), and is independent of the value of the bit means.

Because of the DP noise, we cannot rely on the bit means of unused bits to be zero. Instead, we should apply some filtering to determine which bits are mostly noise, and should have their weight reduced. This is captured in Figure 3a, where we apply a simple heuristic: if the value of a bit mean is less than an absolute threshold, we assume that this bit is capturing noise, and 'squash' it (i.e., we downweigh its importance). The plot shows the effect on RMSE as we vary the threshold, as a multiple of the expected amount of DP noise. It turns out that applying a threshold of 0.1–0.2 is very effective at improving accuracy by almost two orders of magnitude. Figure 3b shows an example in more detail, with a histogram of the estimated bit means for the noisy data with $\epsilon = 3$. We see that the DP noise causes some of these estimates to exceed 1.0 or fall below 0.0 (when the DP subtrahend exceeds the true mean). However, there is a clear "dense" region up to bit 10, with higher bits showing random noise. The bit-squashing approach treats bits 11 and above as noise, and bases the estimate on bits 0-10 only. Figure 3c shows this in practice as we increase the bit depth: the adaptive approach using bit squashing maintains the same level of accuracy, while all other methods grow in error proportional to the magnitude of the (noisy) values.