PREAMBLE: Private and Efficient Aggregation via Block Sparse Vectors

Anonymous Author(s)
Affiliation
Address
email

Abstract

We revisit the problem of secure aggregation of high-dimensional vectors in a two-server system such as Prio. These systems are typically used to aggregate vectors such as gradients in private federated learning, where the aggregate itself is protected via noise addition to ensure differential privacy. Existing approaches require communication scaling with the dimensionality, and thus limit the dimensionality of vectors one can efficiently process in this setup.

We propose PREAMBLE: **Pr**ivate **E**fficient **A**ggregation **M**echanism via **BL**ock-sparse **E**uclidean Vectors. PREAMBLE builds on an extension of distributed point functions that enables communication- and computation-efficient aggregation of *block-sparse vectors*, which are sparse vectors where the non-zero entries occur in a small number of clusters of consecutive coordinates. We show that these block-sparse DPFs can be combined with random sampling and privacy amplification by sampling results, to allow asymptotically optimal privacy-utility trade-offs for vector aggregation, at a fraction of the communication cost. When coupled with recent advances in numerical privacy accounting, our approach incurs a negligible overhead in noise variance, compared to the Gaussian mechanism used with Prio.

1 Introduction

Secure Aggregation is a fundamental primitive in multiparty communication, and underlies several large-scale deployments of federated learning and statistics. Motivated by applications to private federated learning, we study the problem of aggregation of high-dimensional vectors, each of bounded Euclidean norm. We will study this problem in the same trust model as Prio [CB17], where at least one of two servers is assumed to be honest. This setup has been deployed at large scale in practice and our goal in this work is to design algorithms for estimating the sum of a large number of high-dimensional vectors, while keeping the device-to-server communication, as well as the client and server computation small.

This problem was studied in the original Prio paper, as well as in several subsequent works [BBC+19, Tal22, AGJ+22, RSWP22, RU23, ROCT24]. In Prio, the client creates additive secret-shares of its vector and sends those to the two servers. One of the shares can be replaced by a short seed, and thus the communication out of the client for sending D-dimensional vector is $D + O(1)$ field elements. Additionally, such deployments typically require resistance to malicious clients, which can be ensured by sending zero-knowledge proofs showing that the secret-shared vector has bounded norm. Existing protocols [ROCT24] allow for very efficient proofs that incur negligible communication overhead.

In recent years, the size of models that are used on device has significantly increased. Recent work has shown that in some contexts, larger models are easier to train in the private federated learning setup [APF+23, CCT+24]. While approaches have been developed to fine-tuning models while

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

(a) 1-sparse (b) k-sparse

(c) 1-block-sparse (d) k-block-sparse

Figure 1: Possible non-zero patterns (gray) of 1-sparse, k-sparse, 1-block-sparse and k-block-sparse vectors.

training a fraction of the model parameters (e.g. [HSW$^+$22]), increasing model sizes often imply that the number of trained parameters is in the range of millions to hundreds of millions. The communication cost is further exacerbated in the secret-shared setting, as one must communicate D field elements, which are typically 64 or at least 32 bits, even when the gradients themselves may be low-precision. As an example, with a 64-bit field, an eight million-dimensional gradient will require at least 64MB of communication from each client to the servers.

In applications to private federated learning, noise is typically added to the sum of gradients from hundreds of thousands of devices, to provide a provable differential privacy guarantee. Even absent added noise, the aggregate is often inherently noisy in statistical settings. In such setups, computing the exact aggregate may be overkill, and indeed, previous work in the single-trusted-server setting has proposed reducing the communication cost by techniques such as random projections [SYKM17, VBBP$^+$21, VBBP$^+$22, AFN$^+$23, CIK$^+$24]. However, random projections increase the sensitivity of the gradient estimate, and hence naively, would require more noise to be added to protect privacy.

This additional overhead can be reduced if each client picks a different random subset of coordinates, and this subset remain hidden from the adversary. Thus while each client could send a *sparse* vector, the sparsity pattern must remain hidden. This constraint prevents any reduction in communication costs when using Prio. A beautiful line of work [GI14, BGI15, BGI16, BBCG$^+$21] on *Function Secret Sharing* addresses questions of this kind. In particular, *Distributed Point Functions* (DPFs) allow for low-communication secret-sharing of sparse vectors in a high-dimensional space. However, this approach is primarily designed to allow the servers to efficiently compute any one coordinate of the aggregate. The natural extension of their approach to aggregating high-dimensional vectors incurs a non-trivial overheads for the parameter settings of interest, where the sparse vectors have tens of thousands of non-zero entries.

In this work, we address this problem of high-dimensional vector aggregation in a two-server setting. We give the first protocol for this problem that has sub-linear communication cost, reasonable client and server computation costs, and gives near-optimal trade-offs between utility and (differential) privacy. Our approach builds on the ability to efficiently handle the class of *k-block sparse* vectors (see Fig. 1). For a parameter B, we group the D coordinates into $\Delta = D/B$ blocks of B coordinates each. A k-block-sparse vector is one where at most k of these blocks take non-zero values. We make the following contributions:

- We identify block-sparseness as the "right" abstraction, that effectively balances the expressiveness needed for accuracy with the structure needed for efficient cryptographic aggregation protocols.
- We propose an extension of the distributed point function construction of [BGI16] that can secret-share k-block-sparse vectors while communicating $\approx kB$ field elements. For typical parameter settings, our approach that uses blocks is significantly more communication- and computation-efficient, compared to schemes for sparse vectors.
- We show how an aggregation scheme for k-block-sparse inputs combines with sampling analyses for utility, and privacy-amplification-by-sampling analyses for privacy accounting. Combined with recent advances in numerical privacy accounting, we show that for reasonable settings of parameters, our approach leads to privacy-utility trade-offs comparable to the Gaussian mechanism, while providing significantly smaller communication costs. For instance, in the case of an eight million-dimensional vector with 64 bit field size, our approach reduces the communication from $64MB$ to about $1MB$, while increasing the noise standard deviation by about 10% for $(1, 10^{-6})$-DP when aggregating $100K$ vectors.

Overview of Techniques

Compressing high-dimensional vectors for noisy aggregation is a standard consequence of the beautiful advances in randomized sketching algorithms. For example, one can use standard random projection techniques, including efficient versions of the Johnson-Lindenstrauss lemma [JL84, AC06], to construct a sparse unbiased estimator for each vector. A random rotation of the vector, followed by subsampling an appropriate number (say $m \approx 50,000$) of coordinates is sufficient to get a good approximation to the aggregate. Indeed this approach has been proposed for vector aggregation in prior works.

This approach has two significant challenges. Firstly, to get a good approximation for practical parameters, the number of non-zero coordinates m is in the tens of thousands. While sparse vectors can be communicated efficiently using distributed point functions (DPF) constructions in the literature, the overheads in terms of client computation, communication, and server computation are fairly significant. Secondly, this reduction to sparse vectors significantly increases the *sensitivity* of the final estimate and results in larger privacy noise. Indeed for aggregating vectors of norm 1 in $\mathbb{R}^D$, using a γD-sparse vector increases the sensitivity, and thus the required privacy noise, by a multiplicative $1/\gamma$. This leads to an unappealing trade-off between communication and privacy noise.

We show that constraining the sparsity structure to have k non-zero blocks (each of size $B = m/k$) instead of m non-zero coordinates overall can allow for significant performance gains for aggregation protocols in the two-party setting. We further show that this constraint comes at little cost: analyses of privacy amplification by sampling allow us to handle the increased sensitivity with little impact to the privacy-utility tradeoff. We give some details of each of these pieces next.

We show how to efficiently secret-share k-block-sparse vectors. Note that any such vector has at most m non-zero coordinates, and thus is a sum of m 1-sparse vectors. Each of these can be shared using the Distributed Point Function (DPF) construction of [BGI16]. This approach however leads to a communication cost of $m\lambda \log D$, where λ is the security parameter. Thus e.g. when D is more than a million and $\lambda = 128$, the communication cost is at least 300 bytes per non-zero coordinate of the vector. For $m = 50,000$, this amounts to $15MB$ of communication. (This basic scheme also incurs a very significant compute overhead, with the server cost being at least $O(mD)$, though that can be reduced by using optimizations based on probabilistic batch codes [BCGI18] to $O(D + m \log D)$ at a small additional overhead in communication cost.) Thus for m being in the tens of thousands, approaches that only exploit sparsity (rather than *block*-sparsity) are far from feasible.

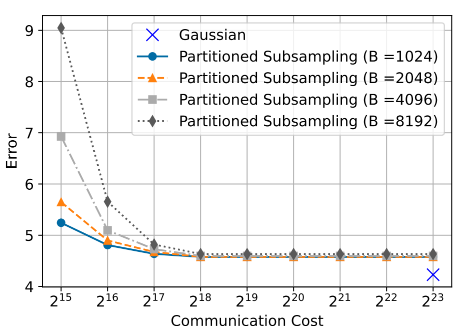


Figure 2: The trade-off between expected squared error and per-client communication, when computing the sum of $n = 10^5$ vectors in $D = 2^{23}$ dimensions with $(1.0, 10^{-6})$-DP. The curves show our algorithm using different block sizes $B \in [10^3, 10^4]$, and the blue 'x' shows the baseline approach of sending the whole vector. We limit ourselves to algorithms for which the server run time is linear or near-linear in D.

We first observe that a simple modification of the DPF construction can exploit the block structure, reducing the communication cost to $O(m + k\lambda \log D)$. This also allows a bulk of the PRG evaluations to be in *counter mode* which can be more computationally efficient. This approach still requires $O(kD)$ server-side PRG evaluations. The use of probabilistic batch codes can reduce this computational overhead. We propose a different approach that uses cuckoo hashing within the DPF construction, that reduces the server's computation to $O(D)$ and reduces by $3\times$ the number of PRG evaluations the server has to do. See Section A for a comparison with prior work and with a concurrent and independent work.

We also provide efficient zero-knowledge proofs of validity for our construction of secret-shared k-block-sparse vectors. This allows the client to prove that all but k of the blocks are zero vectors. The proofs do not change the asymptotic communication or the client (prover) or server (verifiers) runtimes. In particular, the client runtime and communication depend only on the sparsity and on Δ, and not on D. The proof system combines techniques from prior works [BBCG$^+$21] with a novel interactive proof showing that secret shares of PRG seeds in a DPF tree only differ in a small

Aggregate (Informal)

Client Algorithm. Input: vector $v \in \mathbb{R}^D$. Parameters: dimension D, blocksize B, sparsity k.

1. Randomly select $I \subseteq \{1, 2, \ldots, D/B\}$ with $|I| = k$.
2. Define a k-block sparse vector w which is equal to $(\frac{D}{kB}) \cdot v$ in the blocks indexed by I (i.e. coordinates $(j-1)B+1, \ldots, jB$ for $j \in I$) and 0 everywhere else. This rescaling ensures that the expected value of w is v.
3. Use k-block-sparse DPF construction to communicate w to the server. This needs communication $O(kB + k\lambda \log D/B)$.

Servers will decrypt to recover (secret-shares of) each vector w. They collaboratively compute the sum and add noise $\mathcal{N}(0, \sigma^2 \mathbb{I}_d)$.

Figure 3: Informal Description of our approach to Approximate Aggregation via k-block sparse vectors

number of tree nodes (when the seeds are identical the PRG outputs "cancel out", corresponding to a zero block). Finally, we can use an efficient zero-knowledge proof system from prior work [ROCT24] to also prove that that the Euclidean norm of the vector of non-zero values is small.

As discussed above, naively applying random projection techniques to reduce communication results in an unappealing trade-off between communication and privacy noise. We avoid this overhead by exploiting the fact that the sparsity pattern of vectors is hidden from each server in our construction, as in some recent work [CSOK23, CIK$^+$24]. This allows us to use *privacy amplification by sampling* techniques, at a block-by-block level, coupled with composition. For a large range of parameters, it allows us to reduce the overhead in the privacy noise, and asymptotically recover the bounds one would get if we communicated the full vector. Fig. 3 presents an informal outline of our approach. We defer to Section 3 a discussion of different approaches to sampling, and their trade-offs. Coupled with numerical privacy analyses, we get the reduction in communication cost essentially for free (Fig. 2).

While our approach of subsampling blocks rather than coordinates is motivated here for the two-server setting, some of the benefits extend easily to the single trusted server setting where the cost of sending the index would now get amortized over a large block. The privacy analysis of the subsampling approaches requires some control of the norm of each coordinate, which gets relaxed in our approach to a bound on the ℓ_2 norm of each block. The latter is a significantly weaker requirement. Our privacy analysis shows that block-based sampling nearly matches the privacy-utility trade-off of the Gaussian mechanism for a large range of block sizes.

Organization: We start with preliminaries in Section 2 and describe our sampling approach and privacy analysis in Section 3. Our k-block-sparse DPF construction is sketched in Section 4 (with full details in Appendix H). We report our empirical evaluations in Section 5. Additional related work, additional preliminaries, and all proofs are deferred to the Supplement.

2 Preliminaries

We will be working with vectors $v \in \mathbb{R}^D$, and in the linear algebraic parts of the paper, we view them as $v_1, \ldots, v_D$. As is standard, the cryptographic parts of the paper will view the vectors as coming from a finite field $\mathbb{G} = \mathbb{F}_q$; for a suitably large q, the quantized versions of n real vectors can be added without rollover so that we get an estimate of the sum of quantized vectors.

We will use B to represent the block size and $\Delta = D/B$ will be the number of blocks. We will assume Δ is a power of 2 and d will denote $\log_2 \Delta$. λ will denote our security parameter. In the cryptographic part of the paper we will view a vector as a function from $\{0,1\}^d \times [B] \to \mathbb{G}$ where $[B] = \{0, 1, \ldots, B-1\}$.

We recall the definition of (ε, δ)-indistinguishability and differential privacy.

Definition 2.1. *Let $\varepsilon > 0, \delta \in [0, 1)$, and let Y and Y' be two random variables. We say that Y and Y' are (ε, δ)-indistinguishable if for any measurable set S, it is the case that*

$$\Pr[Y' \in S] \leq e^{\varepsilon} \Pr[Y \in S] + \delta$$
$$\text{and } \Pr[Y \in S] \leq e^{\varepsilon} \Pr[Y' \in S] + \delta.$$

Definition 2.2 (Differential Privacy [DMNS06]). *Let $\mathcal{A} : \mathcal{D}^* \to \mathcal{R}$ be a randomized algorithm that maps a dataset $X \in \mathcal{D}^*$ to a range $\mathcal{R}$. We say two datasets X, X' are neighboring if X can be obtained from X' by adding or deleting one element. We say that $\mathcal{A}$ is (ε, δ)-differentially private if for any pair of neighboring datasets X and X', the distributions $\mathcal{A}(X)$ and $\mathcal{A}(X')$ are (ε, δ)-indistinguishable.*

3 Private aggregation via k-block-sparse vectors

In this section we propose two instantiations of our sampling-based sparsification, and describe the formal privacy and utility guarantees of the resulting aggregation algorithms. In both schemes each user is given a vector $v \in \mathbb{R}^D$ which consists of $\Delta = D/B$ blocks, each of size B. We refer to the value of v on block $i \in [\Delta]$ by $v_i \in \mathbb{R}^B$. Our subsampling schemes are parametrized by an upper bound on the ℓ_2 norm of each block L and the number of blocks k to be sent by each user. We will also assume, for simplicity, that k divides Δ. The bound L is somewhat stronger than a total bound on the ℓ_2 norm of the input that is typically assumed in mean estimation. A number of standard techniques are known for converting an ℓ_2 norm bounded vector to an ℓ_∞-bounded vector such as Kashin representation [LV10]. We show that techniques from [AFN$^+$23] can be used to convert a vector of ℓ_2 norm 1 to a vector in which each block has norm $\sqrt{B/D}$ while incurring expected squared error on the order of $1/B$. Note that this result relies crucially on the fact that ensuring that block norms are upper-bounded is easier than ensuring that each coordinate norm is upper-bounded. We formally prove that the expected error due to truncation falls as $\tilde{O}(\frac{1}{B})$ in Appendix F, and evaluate this impact empirically in Section 5.

Our subsampling schemes differ in how the k blocks are subsampled. In **Partitioned Subsampling**, we partition the blocks into k groups of Δ/k consecutive blocks, and pick one block out of each group, randomly and independently. In **Truncated Poisson Subsampling**, we select each block with probability q, to get in expectation $q\Delta$ blocks. If the number of blocks that end up getting subsampled is larger than k, we keep at random k of them. This gives at most k non-zero blocks, which can be communicated using our k-block-sparse DPF. Both schemes give about the same expected utility, and privacy bounds that match asymptotically. The partitioned subsampling results in a more structured k-block-sparse vector, which is a concatenation of k 1-block-sparse vectors, which are simpler to communicate. The truncated Poisson subsampling has more randomness, and can be analyzed numerically using a larger set of tools, though it may have slightly higher communication cost. As we discuss in Section 5, each of these may be preferable over the other for some range of parameters. A formal description of these schemes is given in Fig. 6a and Fig. 6b in the Appendix.

Theorem 3.1 (Utility of the subsampling schemes). *Let $v^1, \ldots, v^n$ be a collection of vectors such that each $v^j = v^j_1, \ldots, v^j_\Delta \in \mathbb{R}^D$ and $\|v^j_i\|_2 \le L$ for all $j \in [n]$ and $i \in [\Delta]$. Let w^j denote the (randomized) report of either of our subsampling algorithms for each $j \in [n]$ and let $W = \sum_{j \in [n]} w^j + N(\bar{0}, \sigma^2 I_D)$ be their noisy aggregate. Then*

$$\mathbf{E}[\|W - \sum_{j \in [n]} v^j\|_2^2] \le nL^2 \frac{\Delta^2}{k} + \sigma^2 \cdot D.$$

Theorem 3.1(proved in Appendix D) provides guidance on how to set the smallest communication cost so that the sub-sampling error is negligible compared to the privacy error. Indeed, assume for simplicity that our vectors lie in the ℓ_∞-ball $v^j \in \left[\frac{-1}{\sqrt{D}}, \frac{1}{\sqrt{D}}\right]^D$. This implies that the norm of each block is upper bounded by $L = \sqrt{B/D}$. Noting that $\Delta = D/B$, the (upper bound above on the) error of the algorithm is $n\Delta/k + \sigma^2 D$. This implies that we should set $kB \ge c \cdot n/\sigma^2$ for some constant $c > 0$. In other words, the number of coordinates that are non-zero must be set to cn/σ^2, which is independent on D, as well as of the blocksize. Thus block-based subsampling does not impact the sampling noise, and any setting of k and B with the product $kB \ge cn/\sigma^2$ would suffice.

We also note that the proof is oblivious to the subsampling method and only uses the fact that marginally, each coordinate has the right expectation, and is non-zero with probability k/Δ. Thus it applies to a range of subsampling methods.

We now analyze the privacy of our subsampling methods using the standard notion of differential privacy [DMNS06] with respect to deletion of user data. In the context of aggregation, we can also think of this adjacency notion as replacing the user's data with the all-0 vector. We state the asymptotic privacy guarantees for our partitioned subsampling method below (the proof can be found in Appendix E). Similar guarantees hold for the Truncated Poisson subsampling with an appropriate choice of parameters and can be derived from the known analyses [ACG+16, Ste22] together with the fact that the truncation operation does not degrade the privacy guarantees [FS25].

Theorem 3.2 (Privacy of partitioned subsampling)**.** *Let $v^1, \ldots, v^n$ be a collection of vectors such that each $v^j = v^j_1, \ldots, v^j_\Delta \in \mathbb{R}^D$ and $\|v^j_i\|_2 \leq L$ for all $j \in [n]$ and $i \in [\Delta]$. Let w^j denote the (randomized) report using Partitioned Subsampling, for each $j \in [n]$ and let A be an algorithm that outputs $W = \sum_{j\in[n]} w^j + N(\bar{0}, \sigma^2 I_D)$. Then there exists a constant c such that for every $\delta > 0$, if $\frac{\sigma}{L} \geq c \cdot \max\left\{\frac{\Delta\sqrt{\log(\Delta/\delta)}}{k}, \sqrt{\frac{\Delta}{k}}\log(\Delta/\delta), \sqrt{\Delta\log(1/\delta)}\right\}$, then A is (ϵ, δ)-differentially private for $\varepsilon = O\left(\frac{L\sqrt{\Delta}\sqrt{\log(1/\delta)}}{\sigma}\right)$.*

Note that this implies that when $L = \sqrt{1/\Delta}$ and we set kB to be n/σ^2, the resulting ε is $O(\sqrt{\log(1/\delta)}/\sigma)$, which asymptotically matches the bound for the Gaussian mechanism. In other words, the privacy cost of our algorithm is close to that of the Gaussian mechanism, when $kB \geq n/\sigma^2$ and $k \geq \sqrt{\Delta \log\frac{1}{\delta}}/\sigma$. For a fixed kB, this constraint translates to an upper bound on the block size.

This asymptotic analysis demonstrates the importance of hiding the sparsity pattern. Specifically, without hiding the pattern we cannot appeal to privacy amplification by subsampling and need to rely on the sensitivity of the aggregated value. The sensitivity is equal to $\sqrt{k} \cdot \frac{L\Delta}{k} = \frac{L\Delta}{\sqrt{k}}$. By the properties of the Gaussian noise addition (Thm. B.2), we obtain that the algorithm is $\left(O\left(\frac{L\Delta\sqrt{\log(1/\delta)}}{\sigma\sqrt{k}}\right), \delta\right)$-DP. This bound is worse by a factor of $\sqrt{\frac{\Delta}{k}}$ than the bound we get in Theorem 3.2. A similar gain was obtained for private aggregation of Poisson subsampled vectors in [CSOK23].

Numerical Privacy Analysis In practice, numerical privacy analysis give much tighter privacy bounds. We can analyze the two approaches described above. For partitioned subsampling, we use the recent work on Privacy Amplification by Random Allocation [FS25] to analyze the privacy cost. The authors show that the Renyi DP parameters of the one-out-of-m version of the Gaussian mechanism can be bounded using numerical methods. Composing this across the k groups gives us the Renyi DP parameters, and hence the overall privacy cost of the full mechanism.

For the approach based on truncated Poisson subsampling, it is shown in [FS25] that the privacy cost is no larger than that of the Poisson subsampling version without the truncation. The Poisson subsampling can then be analyzed using the PRV accountant of [GLW21]. While the PRV accountant usually gives better bounds than one can get from RDP-based accounting, we suffer a multiplicative overhead as the scaling factor κ in Truncated Poisson is smaller than k. For the numerical analysis, we optimize over q to control the overall variance one gets from this process. Note that when k is large and $q \approx \frac{k-O(\sqrt{k})}{\Delta}$, one would expect truncation to be rare, and thus κ to be close to $q\Delta$ and thus to k.

One can also study a different subsampling process where I is a uniformly random subset of size k. We may expect better privacy bounds to hold for this version, intuitively as there is more randomness compared to partitioned subsampling. Feldman and Shenfeld [FS25] show that the privacy bound of this variant is no larger than that of partitioned subsampling. We conjecture that the privacy cost of this version is closer to (untruncated) Poisson with sampling rate $q = k/\Delta$. Since the latter can be more precisely accounted for using the PRV accountant, we expect careful numerical accounting of this version to do better than the RDP-based bounds for partitioned subsampling.

4 k-block-sparse-DPF Construction: Main Ideas

Our construction builds on the tree-based DPF construction from [BGI16] for sharing a 1-sparse function $f : \{0,1\}^d \to \{0,1\}$. f outputs' 0 on all but at most one input $\alpha \in \{0,1\}^d$. At a very high level, in their construction the client shares with each server a seed to a pseudo-random generator. Each server uses their seed to expand out an entire tree with 2^d leaves, where each node in the tree contains the seed to a PRG and some additional "control bits". The client also sends a (public) "correction word" for each layer of the tree. The control bits and the correction word are used to ensure that in each layer, the strings in each node are secret shares of the zero string, except for the single node in that layer that is on the path from the root to the leaf α corresponding to the non-zero output of f.

In our construction of k-sparse DPFs, there are k non-zero nodes in each layer: the nodes that are on the paths from the root to the k non-zero leaves. A naive suggestion could be to also use k correction words for each layer, but this increases the server's computation by a k-fold multiplicative overhead (in a nutshell, each server would need to apply a correction procedure for each node and for each correction word). Instead, we use cuckoo hashing and a collection of slightly more than k correction words per layer, where each node in the layer only needs to apply a correction procedure to two correction words that are relevant to it (the relevant correction words are determined by two hash functions that are public). This reduces the server's computation to 2 corrections per node (the correction procedure is just an XOR: it is quite lightweight). We can also use 3 or 4 hash functions to reduce the number of correction words per layer to be very close to k, see Remark H.3.

To handle *block-sparse* functions we modify the tree: each leaf now corresponds to an entire block (and all but k of them will be zero). The servers expand the seed corresponding to each leaf into an entire block, and the correction words for the last layer ensure that these expansions are secret shares of the correct output (thus the correction words for the last layer are of length B). This avoids the high cost of repeating the bit-output construction B times. Rather than paying a B-multiplicative overhead in the key size, we pay the cost of a single bit-output construction, plus $O(k \cdot B)$ for the correction bits in the last layer. We also only have only a single PRG evaluation per block/leaf (with a large output). This approach reduces the number of large PRG evaluations by a factor of 2-3 compared to techniques from prior work that used cuckoo hashing to construct sparse DPFs (see Section A and Remark H.4). As noted above, if we use 3 or 4 hash functions for cuckoo hashing, then the number of correction words for the last layer is very close to k and the communication complexity approaches $k \cdot B$.

Zero-knowledge proofs of validity. We construct an efficient proof-system that allows a client to prove that it shared a valid block-sparse DPF. The proof is divided into two components:

1. k correction-bit sparse. The client proves that at most k of the (secret shared) correction bits corresponding to leaves in the tree are non-zero.

   These are just bits so here we can use an efficient proof systems of [BGI16] for sharing standard DPFs (the client also needs to send 1-sparse DPFs whose sum is the vector of control bits).

2. k block-sparse. Given that at most k of the correction bits for the leaves are non-zero, the client proves that there are at most k non-zero blocks in the output.

   Consider the final layer of the DPF tree: in a zero block, the two PRG seeds held by the servers are identical (shares of the zero string), whereas in a non-zero block, they are different. Rather than expanding the seeds to B group elements (as in the vanilla construction above), we add another λ bits to the output, and we also add λ corresponding bits to each correction word. We refer to these as the check-bits of the PRG outputs / correction words, and we refer to the original outputs as the payload bits. In the zero blocks the check-bits of the outputs should be identical: subtracting them should results in a zero vector. In each non-zero block, the check-bits of the (appropriate) correction word are chosen so that the correction procedure will result in an all-zero check-bit string. Thus, in our proof system, the servers verify that, in each block, the secret-shared check-bits are indeed all zero. This can be done quite efficiently.

The additional cost for the proof (on top of the construction above) is $O(k \cdot d \cdot \lambda)$ communication, $(k \cdot \text{poly}(d, \lambda))$ client work, $(k \cdot 2^d \cdot \text{poly}(d, \lambda))$ server work for each server. The proof system is sound against a malicious client, but we assume semi-honest behavior by the servers.

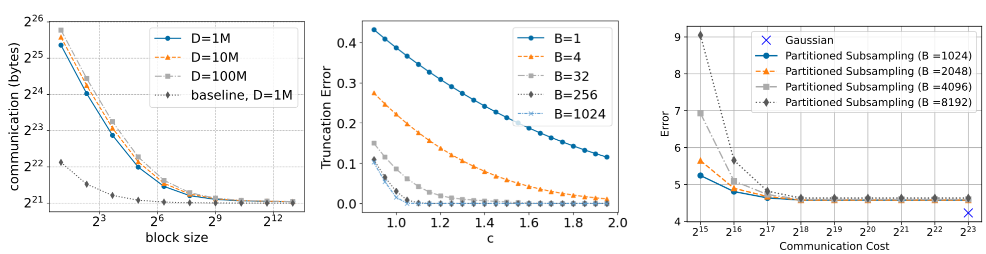
See Section H and A for more details and a comparison with prior and concurrent work.

5 Experimental Evaluation

In this section, we give empirical evidence demonstrating that PREAMBLE is accurate, private, and communication efficient.

Blocking improves communication cost: Figure 4a shows the required communication, or DPF key size, for $\lambda = 128$, a group of size 2^{64} in the final tree layer. Communication depends on the sparsity k, block size B, and data dimension D. We vary B and hold $kB = 2^{18}$ constant. We also include a baseline communication comparison, which is the minimum communication required to communicate kB group elements of size $\log|\mathbb{G}| = \lambda$ and the D/B indices of the non-zero blocks to a single trusted server. Our plots are for 4-way cuckoo hashing, where the cuckoo-hashing overhead is about 1.03. Using fewer hashes would increase the key size, but lead to more efficient server computations.

Blocking reduces Truncation Error We analyze empirically the ease of ensuring a block-wise norm bound. Theoretically, a norm bound of $O(\frac{1}{\sqrt{D}})$ on each entry can be ensured if one transforms to a $O(D)$-dimensional space; the hidden constants in the two $O(\cdot)$ notations are related, where making one smaller makes the other larger. In practice, a simple approach often used is to apply a random rotation to the vector, followed by truncating any entry that is larger than $\frac{c}{\sqrt{D}}$, for an appropriate constant c. For moderate values of c, the *truncation error* (i.e. the norm of the induced error) is small for a random rotation. Imposing the weaker condition that each block has ℓ_2 norm at most $\frac{c}{\sqrt{D/B}}$ results in a lower truncation error. In Fig. 4b, we plot the truncation error as a function of c, when we apply a random rotation, for different values of the block size B. It is easy to see that $c < 1$ will result in non-trivial truncation error for any block size. Our plots show that even moderately large B allow us to take c very close to 1 for a negligible truncation error.

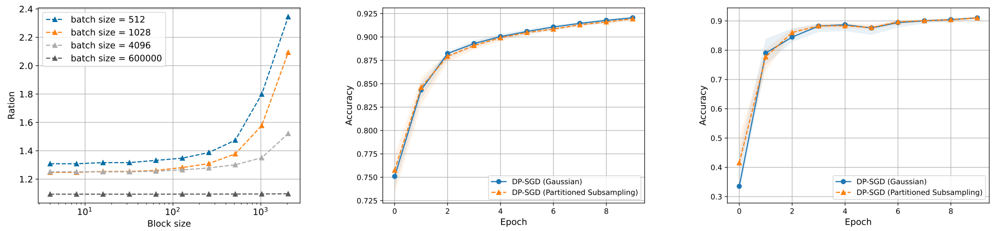


(a) (b) (c)

Figure 4: (Left) Communication vs. Block size for our k-sparse DPF construction, compared to a single-trusted-server baseline. (Middle) The trade-off between the truncation error $\|Trunc_c(Gv) - Gv\|_2$ and constant c, where $v \in \mathbb{R}^D$ is arbitrary, and G is a random rotation matrix. The plots show the error for various block sizes, for $D = 2^{20}$. (Right) The trade-off between the standard deviation of the error and per-client communication $C = kB$, when computing the sum of $n = 10^5$ vectors with dimension $D = 2^{23}$, with $(1.0, 10^{-6})$-DP. The blue 'x' shows the baseline approach of sending the whole vector.

Blocking is compatible with Privacy: We next evaluate the privacy-utility trade-off of our approach, compared to not using any sampling. For our baseline Gaussian mechanism, we use the Analytic Gaussian mechanism analysis from [WBK21]. For these numerical results, we assume that L is fixed to $\sqrt{B/D}$. For each of the approaches, we compute the total variance of the error in the sum, which includes the privacy error that results from the numerical privacy analysis, and the sampling error as bounded by Theorem 3.1. For the communication cost, we simply plot kB as it is a good proxy for the actual communication cost for a large range of parameters. Based on the evaluation above, we consider values of B that are in the range $(2^{10}, 2^{14})$.

Fig. 4c shows the trade-off between communication cost and the standard deviation of the error for our algorithm, using partitioned subsampling, as well as the Gaussian baseline. As is clear from the plots, our approach allows us to significantly reduce the communication costs, at the price of a minor increase in the error. This holds for a range of D from 1M to 8M, and for a range of ε values. (additional plots in the Appendix).



(a) (b) (c)

Figure 5: Comparison between PREAMBLE and the Gaussian mechanism for private model training. (Left) Ratio of the per-batch noise standard deviation of PREAMBLE over the noise of the Gaussian mechanism. (Middle) Accuracy for PREAMBLE and Gaussian mechanism on MNIST with 90% confidence intervals. (Right) Accuracy for PREAMBLE and Gaussian mechanism on CIFAR10 with 90% confidence intervals.

As we decrease the communication kB, the error in Fig. 4c essentially stays constant until a point, and then rapidly increases. This is largely due to the fact that for large block sizes, the norm of each block is larger so that the required lower bound on σ to ensure privacy amplification by subsampling is larger. While the plots are derived from the numerical analysis which is tighter, intuition for this can be derived from the condition $k > \sqrt{\Delta \log 1/\delta}/\sigma$, or equivalently $\sigma > \sqrt{\Delta \log 1/\delta}/k$ in Theorem 3.2. Thus for the case of small kB, one would prefer smaller block sizes. Our plots show that block sizes in the range $[2^{10}, 2^{13}]$ provide low error across a range of parameters. Recall that in Fig. 4a, we saw that block sizes above 2^7 are sufficient to get most of the communication benefits.

Next we turn to experiments simulating the use of PREAMBLE in private model training. In Fig. 5a, we plot the overhead in the per-batch noise standard deviation vs. the block size, if we were to analyse DPSGD [ACG+16] with the Gaussian mechanism for each batch replaced by partitioned subsampling. Due to the more complex accounting that uses general Renyi subsampling, the increase here is larger. Finally, we show some end-to-end experiments for private learning (Fig. 5b and Fig. 5c). We report results for using DP-SGD (with momentum) on MNIST [LCB10] and CIFAR-10 [Kri09]. For MNIST, we use the model from [AFN+23] which has 69050 trainable parameters. For CIFAR, we train a simple two-layer neural network with 66954 parameters on CLIP [RKH+21] embeddings. We give additional details of the setup, including all hyperparameters in Appendix I. Our results show that PREAMBLE allows for significant reduction in communication while incurring a small increase in accuracy. Indeed, for both datasets, PREAMBLE with the chosen parameters allows to communicate around $2 \cdot 10^4$ parameters, compared to roughly $6 \cdot 10^4$ parameters using the Gaussian mechanism. Additional experiments are deferred to the supplement.

6 Conclusions

In this work, we have described PREAMBLE, an efficient algorithm for communicating high-dimensional Euclidean vectors in the Prio model. Our construction reduces this problem to aggregating k-block-sparse vectors, using random sampling, and privacy amplification-by-sampling type analyses to allow private aggregation with a small overhead in accuracy. We showed how to efficiently communicate such vectors, and construct zero-knowledge proofs to validate a bound on the Euclidean norm and k-block-sparsity of these vectors. Our algorithms require client communication proportional to the sparsity kB of these vectors, and our client computation also scales only with kB for parameters of interest. Our construction allows the servers to reconstruct each contribution using $O(D)$ field operations and PRG evaluations in counter mode.

We leave open some natural research directions. Our numerical privacy analyses are close to tight but still have gaps. We conjecture that the k-out-of-Δ sampling approach should admit better privacy analyses than the partitioning-based approach, and it should be no worse than Poisson sampling. Our approach based on Cuckoo hashing with two hash functions incurs a constant factor communication overhead, and has a $O(\frac{1}{k})$ failure probability. While the overhead can be reduced using more hash functions, the failure probability remains k^{-c} for a small constant c [KMW08]. While for our application to approximate aggregation, this has little impact, it would be natural to design a version of our scheme that has a negligible failure probability without increasing the server compute cost.

References

[AC06] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 557–563, New York, NY, USA, 2006. Association for Computing Machinery.

[ACG$^+$16] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 308–318. ACM Press, October 2016.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018.

[AFN$^+$23] Hilal Asi, Vitaly Feldman, Jelani Nelson, Huy Nguyen, and Kunal Talwar. Fast optimal locally private mean estimation via random projections. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 16271–16282. Curran Associates, Inc., 2023.

[AFN$^+$24] Hilal Asi, Vitaly Feldman, Jelani Nelson, Huy L. Nguyen, Kunal Talwar, and Samson Zhou. Private vector mean estimation in the shuffle model: Optimal rates require many messages, 2024.

[AFT22] Hilal Asi, Vitaly Feldman, and Kunal Talwar. Optimal algorithms for mean estimation under local differential privacy. In *International Conference on Machine Learning, ICML, USA*, pages 1046–1056, 2022.

[AG21] Apple and Google. Exposure notification privacy-preserving analytics (ENPA) white paper. `https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf`, 2021.

[AGJ$^+$22] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings*, volume 13409 of *Lecture Notes in Computer Science*, pages 516–539. Springer, 2022.

[APF$^+$23] Sheikh Shams Azam, Martin Pelikan, Vitaly Feldman, Kunal Talwar, Jan Silovsky, and Tatiana Likhomanenko. Federated learning for speech recognition: Revisiting current trends towards large-scale ASR. In *International Workshop on Federated Learning in the Age of Foundation Models in Conjunction with NeurIPS 2023*, 2023.

[AS19] Jayadev Acharya and Ziteng Sun. Communication complexity in locally private distribution estimation and heavy hitters. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, pages 51–60, 2019.

[BBC$^+$19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 67–97. Springer, 2019.

[BBC$^+$23] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Arithmetic sketching. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 171–202. Springer, Cham, August 2023.

[BBCG+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (S & P)*, pages 762–776, 2021.

[BBG20] Borja Balle, Gilles Barthe, and Marco Gaboardi. Privacy profiles and amplification by subsampling. *Journal of Privacy and Confidentiality*, 10(1), 2020.

[BBGN19] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. The privacy blanket of the shuffle model. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Proceedings, Part II*, pages 638–667, 2019.

[BBGN20] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. Private summation in the multi-message shuffle model. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 657–676, 2020.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BDF+18] Abhishek Bhowmick, John C. Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning. *CoRR*, abs/1812.00984, 2018.

[BGH+25] Elette Boyle, Niv Gilboa, Matan Hamilis, Yuval Ishai, and Yaxin Tu. Improved Constructions for Distributed Multi-Point Functions . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2414–2432, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Berlin, Heidelberg, April 2015.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

[BNO08] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 451–468, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BS15] Raef Bassily and Adam D. Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pages 127–135, 2015.

[BS16] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In Martin Hirt and Adam Smith, editors, *Theory of Cryptography*, pages 635–658, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BW18] Borja Balle and Yu-Xiang Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *International Conference on Machine Learning*, 2018.

[CB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In Aditya Akella and Jon Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 259–282. USENIX Association, 2017.

[CCT+24] Geeticka Chauhan, Steve Chien, Om Thakkar, Abhradeep Thakurta, and Arun Narayanan. Training large asr encoders with differential privacy. In *2024 IEEE Spoken Language Technology Workshop (SLT)*, pages 102–109. IEEE, 2024.

[CGH+25] Lynn Chua, Badih Ghazi, Charlie Harrison, Pritish Kamath, Ravi Kumar, Ethan Jacob Leeman, Pasin Manurangsi, Amer Sinha, and Chiyuan Zhang. Balls-and-bins sampling for DP-SGD. In *The 28th International Conference on Artificial Intelligence and Statistics*, 2025.

[CIK+24] Wei-Ning Chen, Berivan Isik, Peter Kairouz, Albert No, Sewoong Oh, and Zheng Xu. Improved communication-privacy trade-offs in l2 mean estimation under streaming differential privacy. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[CJMP22] Albert Cheu, Matthew Joseph, Jieming Mao, and Binghui Peng. Shuffle private stochastic convex optimization. In *The Tenth International Conference on Learning Representations, ICLR*, 2022.

[CKÖ20] Wei-Ning Chen, Peter Kairouz, and Ayfer Özgür. Breaking the communication-privacy-accuracy trilemma. *arXiv preprint arXiv:2007.11707*, 2020.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017.

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(29):1069–1109, 2011.

[CSOK23] Wei-Ning Chen, Dan Song, Ayfer Ozgur, and Peter Kairouz. Privacy amplification via compression: Achieving the optimal privacy-accuracy-communication trade-off in distributed mean estimation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[CSS12] T. H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In Angelos D. Keromytis, editor, *Financial Cryptography and Data Security*, pages 200–214, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[dCP22] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 150–179. Springer, Cham, May / June 2022.

[DJW18] John C. Duchi, Michael I. Jordan, and Martin J. Wainwright. Minimax optimal procedures for locally private estimation. *Journal of the American Statistical Association*, 113(521):182–201, 2018.

[DK12] Michael Drmota and Reinhard Kutzelnigg. A precise analysis of cuckoo hashing. *ACM Trans. Algorithms*, 8(2), April 2012.

[DKM+06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology (EUROCRYPT 2006)*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer Verlag, 2006.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[DR16] Cynthia Dwork and Guy N. Rothblum. Concentrated differential privacy. *CoRR*, abs/1603.01887, 2016.

[DR19] John C. Duchi and Ryan Rogers. Lower bounds for locally private estimation via communication complexity. In *Conference on Learning Theory, COLT*, pages 1161–1191, 2019.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling private contact discovery. *PoPETs*, 2018(4):159–178, October 2018.

[DRS22] Jinshuo Dong, Aaron Roth, and Weijie J. Su. Gaussian differential privacy. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(1):3–37, 02 2022.

[EPK14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 1054–1067, New York, NY, USA, 2014. Association for Computing Machinery.

[FHNP16] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *Journal of Cryptology*, 29(1):115–155, January 2016.

[FM12] Alan Frieze and Páll Melsted. Maximum matchings in random bipartite graphs and the space utilization of cuckoo hash tables. *Random Structures & Algorithms*, 41(3):334–364, 2012.

[FNNT22] Vitaly Feldman, Jelani Nelson, Huy Nguyen, and Kunal Talwar. Private frequency estimation via projective geometry. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 6418–6433. PMLR, 2022.

[FP12] Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp load thresholds for cuckoo hashing. *Random Structures & Algorithms*, 41(3):306–333, 2012.

[FPE16] Giulia Fanti, Vasyl Pihur, and Úlfar Erlingsson. Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries. *Proc. Priv. Enhancing Technol.*, 2016(3):41–61, 2016.

[FS25] Vitaly Feldman and Moshe Shenfeld. Privacy amplification by random allocation, 2025.

[FT21] Vitaly Feldman and Kunal Talwar. Lossless compression of efficient private local randomizers. In Marina Meila and Tong Zhang, editors, *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 3208–3219. PMLR, 2021.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 640–658, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[GKKM22] Badih Ghazi, Pritish Kamath, Ravi Kumar, and Pasin Manurangsi. Faster privacy accounting via evolving discretization. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7470–7483. PMLR, 17–23 Jul 2022.

[GKM+21] Badih Ghazi, Ravi Kumar, Pasin Manurangsi, Rasmus Pagh, and Amer Sinha. Differentially private aggregation in the shuffle model: Almost central accuracy in almost a single message. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, pages 3692–3701, 2021.

[GLW21] Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Numerical composition of differential privacy. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 11631–11642. Curran Associates, Inc., 2021.

[GMPV20] Badih Ghazi, Pasin Manurangsi, Rasmus Pagh, and Ameya Velingker. Private aggregation from fewer anonymous messages. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, pages 798–827, 2020.

[HMR18] Robert Helmer, Anthony Miyaguchi, and Eric Rescorla. Testing privacy-preserving telemetry with prio. `https://hacks.mozilla.org/2018/10/testing-privacy-preserving-telemetry-with-prio/`, 2018.

[HSW+22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[JL84] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into hilbert space. *Contemporary mathematics*, 26:189–206, 1984.

[KH21] Antti Koskela and Antti Honkela. Computing differential privacy guarantees for heterogeneous compositions using fft, 2021.

[KKEPR24] Erki Külaots, Toomas Krips, Hendrik Eerikson, and Pille Pullonen-Raudvere. Slamp-fss: Two-party multi-point function secret sharing from simple linear algebra. Cryptology ePrint Archive, Report 2024/1394, 2024. `https://eprint.iacr.org/2024/1394.pdf`.

[KLN+08] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 531–540, 2008.

[KMW08] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. In Dan Halperin and Kurt Mehlhorn, editors, *Algorithms - ESA 2008*, pages 611–622, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[KOV15] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Secure multi-party differential privacy. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2008–2016. Curran Associates, Inc., 2015.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[LV10] Y. Lyubarskii and R. Vershynin. Uncertainty principles and vector quantization. *Information Theory, IEEE Transactions on*, 56(7):3491–3501, 2010.

[MM18] Sebastian Meiser and Esfandiar Mohammadi. Tight on budget? tight bounds for r-fold approximate differential privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 247–264, New York, NY, USA, 2018. Association for Computing Machinery.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 1273–1282, 2017.

[MTZ19] Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi differential privacy of the sampled gaussian mechanism. *ArXiv*, abs/1908.10530, 2019.

[NXY+16] Thông T. Nguyên, Xiaokui Xiao, Yin Yang, Siu Cheung Hui, Hyejin Shin, and Junbum Shin. Collecting and analyzing data from smart device users with local differential privacy. *CoRR*, abs/1606.05053, 2016.

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Proceedings of the 9th Annual European Symposium on Algorithms*, ESA '01, page 121–133, Berlin, Heidelberg, 2001. Springer-Verlag.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, August 2014.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 18–24 Jul 2021. Model at `https://github.com/openai/CLIP/`.

[ROCT24] Guy N. Rothblum, Eran Omri, Junye Chen, and Kunal Talwar. PINE: Efficient verification of a euclidean norm bound of a Secret-Shared vector. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6975–6992, Philadelphia, PA, August 2024. USENIX Association.

[RSWP22] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. ELSA: secure aggregation for federated learning with malicious actors. *IACR Cryptol. ePrint Arch.*, page 1695, 2022.

[RU23] Olivia Röhrig and Maxim Urschumzew. `dpsa4fl`: Differential privacy for federated machine learning with PRIO, 2023.

[SFZ+14] Chongjing Sun, Yan Fu, Junlin Zhou, Hui Gao, et al. Personalized privacy-preserving frequent itemset mining using randomized response. *The Scientific World Journal*, 2014, 2014.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

[SMM19] David Sommer, Sebastian Meiser, and Esfandiar Mohammadi. Privacy loss classes: The central limit theorem in differential privacy. *Proceedings on Privacy Enhancing Technologies*, 2019:245–269, 04 2019.

[Ste22] Thomas Steinke. Composition of differential privacy & privacy amplification by subsampling, 2022.

[SYKM17] Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, and H. Brendan McMahan. Distributed mean estimation with limited communication. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[Tal22] Kunal Talwar. Differential secrecy for distributed data and applications to robust differentially secure vector summation. In L. Elisa Celis, editor, *3rd Symposium on Foundations of Responsible Computing, FORC 2022, June 6-8, 2022, Cambridge, MA, USA*, volume 218 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[TWM+24] Kunal Talwar, Shan Wang, Audra McMillan, Vitaly Feldman, Pansy Bansal, Bailey Basile, Aine Cahill, Yi Sheng Chan, Mike Chatzidakis, Junye Chen, Oliver R. A. Chick, Mona Chitnis, Suman Ganta, Yusuf Goren, Filip Granqvist, Kristine Guo, Frederic Jacobs, Omid Javidbakht, Albert Liu, Richard Low, Dan Mascenik, Steve Myers, David Park, Wonhee Park, Gianni Parsa, Tommy Pauly, Christian Priebe, Rehan Rishi, Guy N. Rothblum, Congzheng Song, Linmao Song, Karl Tarbe, Sebastian Vogt, Shundong Zhou, Vojta Jina, Michael Scaria, and Luke Winstrom. Samplable anonymous aggregation for private federated data analysis. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 2859–2873, New York, NY, USA, 2024. Association for Computing Machinery.

[VBBP+21] Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. Drive: One-bit distributed mean estimation. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 362–377. Curran Associates, Inc., 2021.

[VBBP+22] Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben Itzhak, and Michael Mitzenmacher. Eden: Communication-efficient and robust distributed mean estimation for federated learning. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

[WBK21] Yu-Xiang Wang, Borja Balle, and Shiva Kasiviswanathan. Subsampled rényi differential privacy and analytical moments accountant. *Journal of Privacy and Confidentiality*, 10(2), 2021.

[YB18] Min Ye and Alexander Barg. Optimal schemes for discrete distribution estimation under locally differential privacy. *IEEE Trans. Inf. Theory*, 64(8):5662–5676, 2018.

[ZDW22] Yuqing Zhu, Jinshuo Dong, and Yu-Xiang Wang. Optimal accounting of differential privacy via characteristic function. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 4782–4817. PMLR, 28–30 Mar 2022.

[ZWC+22] Mingxun Zhou, Tianhao Wang, T.-H. Hubert Chan, Giulia Fanti, and Elaine Shi. Locally differentially private sparse vector aggregation. In *43rd IEEE Symposium on Security and Privacy, SP*, pages 422–439, 2022.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction summarize our contributions, that are detailed in the rest of the paper and the Supplementary material.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in the conclusions section, where they fit better with the open questions.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All theorems state their assumptions, and are proven in the main paper or the supplement.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Our algorithms are described in full detail. All experiments are detailed in the text, or in the appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Our main contribution is algorithmic, and the algorithm is described with very detailed pseudocode. The experiments in the paper are fully described. We evaluate on standard ML benchmarks.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All details are described in the paper or the supplement.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We provide error bars for our experiments on real data.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: we describe the computational setup of our algorithms in the supplement.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper conforms to the NeurIPs code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: This paper presents work whose goal is to advance the field of private and secure data analysis. There are many potential societal consequences of our work, none which we feel must be specifically highlighted.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: the paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: the paper only uses standard benchmarks and acknowledges/cites them.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: the paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: the paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: the paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: the core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Additional Related Work

As discussed above, Prio was introduced in [CB17], and triggered a long line of research on efficient validity proofs for various predicates of interest. Perhaps most related to our work is POPLAR [BBCG$^+$21], which uses Distributed Point Functions (DPFs) to allow clients to communicate 1-sparse vectors in a high-dimensional space. They are used in that work to solve heavy hitters over a large alphabet, where the servers together run a protocol to compute the heavy hitters. In their work, the aggregate vectors is never constructed, and hence the DPFs there are optimized for *query* access: at any step, the servers want compute a specific entry of the aggregate vector. In contrast, we are interested in the setting when the whole aggregate vector is needed, and this leads to different trade-offs in our use of DPFs.

We consider not just single-point DPFs, as used in POPLAR, but rather a generalization for a larger number of non-zero evaluations. [BCGI18] also consider such multi-point functions and improve upon the naive implementation of a k-sparse DPF using k single-point DPF instantiations. Multi-point function secret sharing has also been optimized using cuckoo hashing [ACLS18, SGRR19, dCP22]. They use cuckoo hashing to break one vector with k non-zero entries into $k' > k$ smaller vectors with one non-zero entry each, after which standard DPF keys are constructed for those k' vectors. Applying cuckoo hashing in this way incurs a roughly $2\times$-$3\times$ overhead in the total number of vector entries, dependent on the number of hash functions chosen and the cuckoo hashing memory overhead. While these prior work didn't focus on block-sparse DPFs, applying their methodlogy to block-sparse functions would result in a similar $2\times$-$3\times$ overhead in terms of the total number of blocks and thus also the total number of large PRG evaluations (with output length B). Our construction applies cuckoo hashing differently and avoids this computational overhead. Finally, we remark that similar cuckoo hashing-based approaches [CLR17, DRRT18, FHNP16, PSZ14] have also been used in the context of private set intersection.

Another recent work in this line of research [BBC$^+$23] uses projections, which may at first seem related to our work. Unlike our work, these arithmetic sketches are designed for the servers to be able to verify certain properties efficiently, without any help from the client. In contrast, our use of sketching is extraneous to the cryptographic protocol, and helps us reduce the client to server communication.

The fact that differential privacy guarantees get amplified when the mechanism is run on a random subsample (that stays hidden) was first shown by Kasiviswanathan et al. [KLN$^+$08] and has come to be known as privacy amplification by sampling. Abadi et al. [ACG$^+$16] first showed that careful privacy accounting tracking the moments of the privacy loss random variable, and numerical privacy accounting techniques can provide significantly better privacy bounds. In effect, this approach tracks the Renyi DP parameters [MTZ19] or Concentrated DP [DR16, BS16] parameters. Subsequent works have further improved numerical accounting techniques for the Gaussian mechanism [BW18] and for various subsampling methods [WBK21, BBG20]. Tighter bounds on composition of mechanisms can be computed by more carefully tracking the distribution of the privacy loss random variable, and a beautiful line of work [KOV15, DRS22, MM18, SMM19, KH21, GLW21, GKKM22, ZDW22]. Numerical accounting for privacy amplification for a specific sampling technique we use has been studied recently in [CGH$^+$25, FS25].

The Prio architecture has been deployed at scale for several applications, including by Mozilla for private telemetry measurement [HMR18] and by several parties to enable private measurements of pandemic data in Exposure Notification Private Analytics (ENPA) [AG21]. Talwar et al. [TWM$^+$24] show how Prio can be combined with other primitives to build an aggregation system for differentially private computations.

We remark that the problem of vector aggregation has attracted a lot of attention in different models of differential privacy. Vector aggregation is a crucial primitive for several applications, such as deep learning in the federated setting [ACG$^+$16, MMR$^+$17], frequent itemset mining [SFZ$^+$14], linear regression [NXY$^+$16], and stochastic optimization [CMS11, CJMP22]. The privacy-accuracy trade-offs for vector aggregation are well-understood for central DP [DKM$^+$06], for local DP [BNO08, CSS12, DR19, DJW18, BDF$^+$18, AFT22], as well as for shuffle DP [BBGN19, BBGN20, GMPV20, GKM$^+$21]. Several works have addressed the question of reducing the communication cost of private aggregation, in the local model [CKÖ20, FT21, AFN$^+$23], and under sparsity assumptions [BS15, FPE16, YB18, AS19, ZWC$^+$22]. While the shuffle model can allow for accurate vector aggregation [GKM+21], recent work [AFN+24] has shown that for large D, the number of messages per client must be very large, thus motivating an aggregation functionality.

Our use of subsampling to reduce communication in private vector aggregation is closely related to work on aggregation in the single trusted server, secure multi-party aggregation and multi-message shuffling models [CSOK23, CIK+24]. Aside from a different trust model, our work crucially relies on blocking to reduce the sparsity. Sparsity constraints also make regular Poisson subsampling less suitable for our application and necessitate the use of sampling schemes that require a more involved privacy analysis.

Very recent works. After a preliminary version of our work was made public, we were made aware of an independent and concurrent work [BGH+25] that studies the problem of communicating m-sparse vectors in the two-server setting. They compare three different schemes for this task and empirically compare them. Their "big-state" DMPF is equivalent to our naive scheme and their probabilistic batch codes (PBC) construction uses cuckoo hashing in a black-box way, similarly to the prior works discussed above. They also propose a new scheme based on Oblivious Key-value stores (OKVS), which obtains constant-factor improvements in the server runtime compared to the other two schemes for some range of sparsity. To keep the server-side computation low, the OKVS-based schemes (which allow a range of tradeoff) incur at least a $2\times$ communication overhead.

Beyond these differences in various performance measures, one of our main contributions is identifying block-sparsity as a property that lends itself both to significant savings in DPF constructions and to efficient privacy-preserving aggregation. [BGH+25] don't focus on block-sparse DPFs, but their approaches can be applied towards block-sparse constructions. The cuckoo-hashing based constructions would incur a $3\times$ overhead in the number of large PRG evaluations (as discussed above). The OKVS-based schemes incur the above $2\times$ communication overhead.

Another recent work [KKEPR24] also provides an efficient multi-point DPF construction (they also don't focus on block-sparsity). Their scheme requires the client to solve linear systems with k equations and at least $(k + \lambda)$ unknowns over a field of size 2^λ. So even if the blocks are small, for large k the client work is larger than in our scheme.

B Additional Preliminaries

Definition B.1. *Function Secret Sharing [BGI15] Let $\mathcal{F} = \{f : I \to \mathbf{G}\}$ be a class of functions with input domain I and output group $\mathbb{G}$, and let $\lambda \in \mathbb{N}$ be a security parameter. A 2-party function secret sharing (FSS) scheme is a pair of algorithms with the following syntax:*

- *$Gen(1^\lambda, f)$ is a probabilistic, polynomial-time key generation algorithm, which on input 1^λ and a description of a function f outputs a tuple of keys (k_0, k_1).*
- *$Eval(i, k_i, x)$ is a polynomial-time evaluation algorithm, which on input server index $i \in \{0, 1\}$, key k_i, and input $x \in I$, outputs a group element $y \in \mathbb{G}$.*

Given some allowable leakage function $Leak : \{0, 1\}^* \to \{0, 1\}^*$ and a parameter $\gamma \in [0, 1]$, we require the following two properties:

- *Correctness: For any $f \in \mathcal{F}$ and any $x \in I$, we have that*
$$Pr[\sum_{b\in\{0,1\}} Eval(b, k_b, x) = f(x)] \geq 1 - \gamma.$$
- *Security: For any $b \in \{0, 1\}$, there exists a ppt simulator such that for any polynomial-size function $f \in \mathcal{F}$:*
$$\{k_b|(k_0, k_1) \leftarrow Gen(1^\lambda, f)\} \sim \{k_b \leftarrow Sim_b(1^\lambda, Leak_b(f))\}$$

Note that we have defined a relaxed notion of correctness here, where we allow a small probability γ of incorrect evaluation. While γ will be 0 for some of our constructions, the most efficient version of our protocol will have a γ that is polynomially small in the sparsity k.

We define 1-sparse and k-block-sparse distributed point functions, which are instantiations of FSS for specific families of sparse functions:

Definition B.2 (Distributed Point Function (DPF)). *A point function $f_{\alpha,\beta}$ for $\alpha \in \{0,1\}^d$ and $\beta \in \mathbb{G}$ is defined to be the function $f : \{0,1\}^d \to \mathbb{G}$ such that $f(\alpha) = \beta$ and $f(x) = 0$ for $x \neq \alpha$. A DPF is an FSS for the family of all point functions.*

Definition B.3 (k-block-sparse DPF). *A k-block-sparse function $f_{\alpha,\beta}$ with block size B for $\alpha = \{\alpha^0, \dots, \alpha^{k-1}\}$, where $\alpha^i \in \{0,1\}^d$ and $\beta = \{\beta^0, \dots, \beta^{k-1}\}$ and $\beta^i = \{\beta^i_0, \dots, \beta^i_{B-1}\} \in \mathbb{G}^B$ is defined to be the function $f : \{0,1\}^{d+\log B} \to \mathbb{G}$ such that $f(\alpha^i||j) = \beta^i_j$ and $f(x) = 0$ for $x \neq \alpha^i||j$ with $j \in [B]$ and $i \in [k]$. A k-block-sparse DPF is an FSS for the family of all k-block-sparse functions.*

Cuckoo Hashing. Cuckoo Hashing [PR01] is an algorithm for building hash tables with worst case constant lookup time. The hash table depends on two (or more) hash functions, and each item is placed in location specified by one of the hash values. When inserting k items from a universe U into a cuckoo hash table of size $\tilde{k}$, the hash functions map U to $[\tilde{k}]$. When the hash functions are truly random, as long as k is a constant fraction of $\tilde{k}$, there is a way to assign each item to one of its hash locations while avoiding any collision:

Theorem B.1. *Cuckoo Hashing [DK12] Suppose that $c \in \{0,1\}$ is fixed. The probability that a cuckoo hash of $k = \lfloor (1-c)\tilde{k} \rfloor$ data points into two tables of size $\tilde{k}$ succeeds is equal to:*

$$1 - \frac{(2c^2 - 5c + 5)(1-c)^3}{12(2-c)^2 c^3} \frac{1}{\tilde{k}} + \mathcal{O}\left(\frac{1}{\tilde{k}^2}\right)$$

Note that if $c \approx 0.32$, this simplifies to about $1 - 1/\tilde{k} - \mathcal{O}(1/\tilde{k}^2)$. Therefore, this choice of c leads to a failure probability of $\tilde{\mathcal{O}}(1/\tilde{k})$. Variants of cuckoo hashing where more than 2 hash functions are used can improve the space efficiency of the data structure. E.g. using 4 hash functions allows for a an efficiency close to 0.97 with probability approaching 1 [FP12, FM12].

We will use the following standard result in differential privacy.

Theorem B.2 (Gaussian Mechanism [DKM+06, DR14]). *Let $\varepsilon, \delta \in (0,1)$. Let $\mathcal{A} : (\mathbb{R}^D)^* \to \mathbb{R}^D$ be the mechanism that for a sequence of vectors $v_1, \dots, v_n \in \mathbb{R}^D$ outputs $\sum_i v_i + \mathcal{N}(0, \sigma^2 \mathbb{I}_D)$. If for all $i \in [n]$, the input v_i is restricted to $\|v_i\|_2 \leq s$ and $(\sigma/s)^2 \geq 2\log(1.25/\delta)/\varepsilon^2$, then $\mathcal{A}$ is (ε, δ)-differentially private. We refer to σ as the scale of the mechanism. In this context s is the sensitivity of the sum to adding/deleting an element.*

C Pseudocode for sampling schemes

D Missing Proofs from Section 3

Proof of Theorem 3.1: First, note that $\mathbf{E}[w^j] = v^j$ for all $j \in [n]$. Therefore we have

$$\begin{aligned}
\mathbf{E}[\|W - \sum_{j\in[n]} v^j\|_2^2] &= \mathbf{E}\left[\|\sum_{j\in[n]} w^j - v^j + N(\bar{0}, \sigma^2 I_D)\|_2^2\right] \\
&= \sum_{j=1}^{n} \mathbf{E}\left[\|w^j - v^j\|_2^2\right] + \sigma^2 D \\
&= n\mathbf{E}\left[\|w^1 - v^1\|_2^2\right] + \sigma^2 D \\
&\leq n\Delta \tfrac{k}{\Delta}(1 - \tfrac{k}{\Delta}) L^2 (\frac{\Delta}{k})^2 + \sigma^2 D \\
&\leq n \cdot L^2(\Delta^2/k) + \sigma^2 D.
\end{aligned}$$

Here we have used the fact that the variance of the w^1 decomposes across Δ blocks, where each block contributed $p(1-p)$ times the square of its value when non-zero (which is $L(\Delta/k)$) and $p = \frac{k}{\Delta}$ is the probability of a block being non-zero. $\square$

SampledVector (Partitioned Subsampling)

Input: vector $v \in \mathbb{R}^D$. Parameters: dimension D, blocksize B, sparsity k, sampling probability q, $\Delta = D/B$, blockwise ℓ_2 bound L.

1. Split the block indices into k consecutive subsets $S_j = \{(j-1)\Delta/k+1, \ldots, j\Delta/k\}$ for $j \in [k]$.
2. Select an index i_j randomly and uniformly from S_j and define $I = \{i_j\}_{j\in[k]}$
3. Define the subsampled (and clipped) v^I as
$$v_i^I = \begin{cases} \frac{\Delta}{k} \cdot \mathrm{clip}_L(v_i) & \text{if } i \in I \\ 0 & \text{otherwise} \end{cases},$$
where $\mathrm{clip}_L(z)$ is defined as z if $\|z\|_2 \leq L$ and $\frac{L}{\|z\|_2} \cdot z$, otherwise.
4. Output $w = v^I$.

(a) k-wise 1-sparse sampling scheme

SampledVector (Truncated Poisson Subsampling)

Input: vector $v \in \mathbb{R}^D$. Parameters: dimension D, blocksize B, sparsity k, sampling probability q, $\Delta = D/B$, blockwise ℓ_2 bound L.

1. Select a subset I_0 by picking each coordinate in $\{1, \ldots, \Delta\}$ independently with probability q.
2. If $|I_0| > k$, let I be random subset of I_0 of size k. Else set $I = I_0$.
3. Let $\kappa = \mathbb{E}[|I|] = \mathbb{E}[\min(\mathtt{Bin}(\Delta, q), k)]$ under this sampling process.
4. Define the subsampled (and clipped) v^I as
$$v_i^I = \begin{cases} \frac{\Delta}{\kappa} \cdot \mathrm{clip}_L(v_i) & \text{if } i \in I \\ 0 & \text{otherwise} \end{cases}.$$
5. Output $w = v^I$.

(b) k-sparse sampling scheme

Figure 6: Pseudocode for Subsampling algorithms.

E Proof of Theorem 3.2

We will need the following asymptotic bound on the privacy of Poisson subsampling of the Gaussian noise addition.

Lemma E.1 ([ACG+16]). *Let $A_1, \ldots, A_T : (\mathbb{R}^B)^n \to \mathbb{R}^B$ be a sequence of Gaussian noise addition algorithms with sensitivity s and noise scale σ. Let $P_\eta(A_1, \ldots, A_T)$ be the Poisson subsampling scheme in which each user's data is used in each step with probability η randomly and independently (of other users and steps). Then, there exist a constant c such that for every $\delta > 0$, if $\sigma/s \geq c\eta\sqrt{T\log(1/\delta)}$ then $P_\eta(A_1, \ldots, A_T)$ satisfies (ε, δ)-DP for $\varepsilon = O(\eta\frac{s}{\sigma}\sqrt{T\log(1/\delta)})$.*

of Theorem 3.2. We first observe that we can analyze algorithm as an independent composition of $A_1, \ldots, A_k$, with the instance A_j outputting the coordinates of W in the set $S_j = \{(j-1)\Delta/k + 1, \ldots, j\Delta/k\}$ for $j \in [k]$. For convenience of notation we will analyze A_1 (with the rest being identical). Observe that the output of A_1 is $W_1, \ldots, W_{\Delta/k}$. Now, by the definition of the sampling scheme any user's data is summed in exactly one (uniformly chosen) of $W_1, \ldots, W_{\Delta/k}$. Each of these algorithms is Gaussian noise addition with sensitivity $L\Delta/k$ and scale σ. This implies that we can apply results for privacy amplification by allocation for Gaussian noise from [FS25] to analyze this algorithm. For the analytic results we use an upper bound on the privacy parameters of random allocation in terms of Poisson subsampling for Gaussian noise. Specifically, for every ε, k-wise composition of 1 out of Δ/k random allocation for Gaussian noise addition algorithms satisfies $(\varepsilon, \delta_P + \Delta\delta_0 + \delta')$-DP, where (ε, δ_P) are the privacy parameters of Δ-step Poisson subsampling scheme with rate $\eta = \frac{k}{\Delta(1-\gamma)}$ for $\gamma = (e^{\varepsilon_0} - e^{\varepsilon_0})\sqrt{\frac{k}{2\Delta}\ln\left(\frac{k}{\delta'}\right)}$. Here $(\varepsilon_0, \delta_0)$ are the privacy parameters of each Gaussian noise addition. Note that the sensitivity of the aggregate in each block is $L\Delta/k$. Therefore, by setting $\delta_0 = \delta/(3\Delta)$ we get $\varepsilon_0 = \frac{L\Delta\sqrt{2\ln(15\Delta/(4\delta))}}{k\sigma}$ (Thm. B.2). We set $\delta' = \delta/3$ and note that by the first part of our assumption on σ/L, $\varepsilon_0 \leq 1$ and therefore $e^{\varepsilon_0} - e^{\varepsilon_0} \leq 3\varepsilon_0$. Now the second part of our assumption of σ/L implies that

$$\begin{aligned}\gamma &\leq 3\varepsilon_0\sqrt{\frac{k}{2\Delta}\ln\left(\frac{3k}{\delta}\right)} \\ &\leq \frac{3L\Delta\sqrt{2\ln(15\Delta/(4\delta))}}{k\sigma}\sqrt{\frac{k}{2\Delta}\ln\left(\frac{3k}{\delta}\right)} \\ &\leq \frac{3L\sqrt{\Delta}\ln(15\Delta/(4\delta))}{\sqrt{k}\sigma} \leq 1/2.\end{aligned}$$

This implies that $\eta \leq \frac{2k}{\Delta}$. Now, by Lemma E.1, the Δ-step Poisson subsampling scheme with subsampling rate η is $(\varepsilon, \delta/3)$-DP for

$$\varepsilon = O\left(\frac{L\sqrt{\Delta}\sqrt{\log(1/\delta)}}{\sigma}\right).$$

Here we note that the conditions of the lemma translate to

$$\frac{\sigma k}{L\Delta} \geq c\frac{k}{\Delta}\sqrt{\Delta\log(1/\delta)}$$

or equivalently, $\frac{\sigma}{L} \geq c\sqrt{\Delta\log(1/\delta)}$ (which is ensured by the third part of our assumption). Finally, noting that $\Delta\delta_0 + \delta' \leq 2\delta/3$, we get the claimed bound. ☐

F Ensuring block-wise norm bound

We now formally show that one can reduce the problem of computing means of ℓ_2 bounded vectors to our setting of block-wise bounded norm. Without loss of generality, we can assume that each input vector has ℓ_2 norm 1. Our main reduction is based on the techniques developed by Asi *et al.* [AFN+23] in the context of communication-efficient algorithms for mean estimation with local differential privacy. Their randomizer for mean estimation relies on a randomized dimensionality reduction followed by an optimal differentially private randomizer in lower dimension referred to as `PrivUnit`. `PrivUnit` requires a vector of unit length as an input, whereas the randomized dimensionality reductions they use result in vectors of varying lengths. Asi *et al.* apply scaling to ensure that the norm condition is satisfied and develop several techniques for the analysis of the error resulting from this step. We observe that their dimensionality reductions can be used just as (randomized) linear maps (in the same dimension) with each block of B coordinates in the image corresponding to the projection of the original input into B dimensions. Thus we can apply clipping/scaling to ensure that block norms are upper bounded and then analyze the resulting error in essentially the same way as in [AFN+23]. Our first application of this approach shows that a random rotation with simple block norm clipping to $\sqrt{B/D}$ achieves expected squared error of $1/B$.

Theorem F.1. *For a vector $v = v_1, \ldots, v_\Delta \in \mathbb{R}^D$, where $v_i \in \mathbb{R}^B$ let $\mathtt{blkclip}_B(v)$ denote the vector $u = u_1, \ldots, u_\Delta$, such that for every $i \in [\Delta]$, $u_i = clip_{\sqrt{B/D}}(v_i)$. Let $U \in \mathbb{R}^{D\times D}$ be a randomly and uniformly chosen rotation matrix. Then for every $v \in \mathbb{R}^D$ such that $\|v\|_2 \leq 1$,*

$$\mathop{\mathbb{E}}_{U}\left[\left\|U^\top \mathtt{blkclip}_B(Uv) - v\right\|_2^2\right] = O\left(\frac{1}{B}\right).$$

Our proof of this result relies on the following lemma from [AFN+23].

Lemma F.2 ([AFN+23])**.** *Let x be a random unit vector on the unit ball of $\mathbb{R}^D$ and z be the projection of x onto the last B coordinates. We have*

$$\left|\mathbb{E}[\|z\|_2] - \sqrt{B/D}\right| = O\left(\frac{1}{\sqrt{BD}}\right)$$

Proof of Theorem F.1. We first note that the squared error scales quadratically with the norm of v and therefore it is sufficient to prove the theorem for unit norm v. For a given U, Let $w = Uv$ and $w_1, \ldots, w_\Delta$ be the blocks of size B in w. Observe that when U is a randomly chosen rotation matrix, w is a random and uniform unit vector. Naturally, the uniform distribution over random unit vectors is not affected by permuting coordinates and therefore for every $i \in [\Delta]$ we can apply Lemma F.2 to get

$$\left| \mathop{\mathbb{E}}_{U}[\|w_i\|_2] - \sqrt{B/D} \right| = O\left(\frac{1}{\sqrt{BD}}\right).$$

In addition, by the same symmetry, we have that

$$\mathop{\mathbb{E}}_{U}[\|w_i\|^2] = \frac{B}{D}.$$

Combining these two results, we have

$$\begin{aligned}
\mathop{\mathbb{E}}_{U}\left[\left\|U^\top \texttt{blkclip}_B(Uv) - v\right\|_2^2\right] &= \mathop{\mathbb{E}}_{U}\left[\left\|UU^\top \texttt{blkclip}_B(Uv) - Uv\right\|_2^2\right] \\
&= \mathop{\mathbb{E}}_{U}\left[\|\texttt{blkclip}_B(w) - w\|_2^2\right] \\
&= \sum_{i \in [\Delta]} \mathop{\mathbb{E}}_{U}\left[\left\|\text{clip}_{\sqrt{B/D}}(w_i) - w_i\right\|_2^2\right] \\
&\leq \sum_{i \in [\Delta]} \mathop{\mathbb{E}}_{U}\left[\left(\|w_i\|_2 - \sqrt{B/D}\right)_2^2\right] \\
&= \sum_{i \in [\Delta]} \mathop{\mathbb{E}}_{U}\left[\left(2\frac{B}{D} - 2\sqrt{B/D}\|w_i\|_2\right)\right] \\
&= 2\sqrt{B/D} \cdot \sum_{i \in [\Delta]} \mathop{\mathbb{E}}_{U}\left[\left(\sqrt{B/D} - \|w_i\|_2\right)\right] \\
&= O\left(\frac{D}{B} \cdot \sqrt{B/D} \cdot \frac{1}{\sqrt{BD}}\right) = O(1/B)\,.
\end{aligned}$$

□

While this method is simple to describe and analyze, it is relatively inefficient as it requires D^2 multiplications. We also show that a significantly more efficient scheme from [AFN+23] based on Subsampled Randomized Hadamard Transform (SHRT) can also be easily adapted to our setting at the expense of somewhat worse $\tilde{O}(1/\sqrt{B})$ expected squared error. Specifically, let $W = SHT$ denote the following distribution over random matrices: $H \in \mathbb{R}^{D \times D}$ is the Hadamard matrix, $S \in \mathbb{R}^{D \times D}$ is a random permutation matrix, and $T \in \mathbb{R}^{D \times D}$ is a diagonal matrix where $T_{i,i}$ are independent samples from the Rademacher distribution (that is, uniform over ± 1). An important (and well-known) property of this family of matrices is that multiplication by W and $W^\top$ can be performed in time $O(D \log D)$ [AC06].

Theorem F.3. *Let $W \in \mathbb{R}^{D \times D}$ be a randomly and uniformly chosen SHT matrix as described above. Then for every $v \in \mathbb{R}^D$ such that $\|v\|_2 \leq 1$,*

$$\mathop{\mathbb{E}}_{W}\left[\left\|W^\top \texttt{blkclip}_B(Wv) - v\right\|_2^2\right] = O\left(\frac{\log^2 D}{B}\right).$$

Further, multiplication by W and $W^\top$ can be performed in time $O(D \log D)$.

To prove this result, we first establish some relevant properties of SHRT.

Lemma F.4 ([AFN+23])**.** *Suppose $W_B = S_B HT$ is obtained with S_B being a $B \times D$ uniform sampling matrix without replacement, H being Hadamard matrix and T being a Rademacher diagonal matrix as above. Then for some constant $C > 0$, for any fixed $u \in \mathbb{R}^D$ of unit Euclidean norm and $\delta \in (0, 1)$,*

$$\Pr_{W_B}\left[\left|\|W_B u\|_2^2 - \sqrt{\frac{B}{D}}\right| > C\sqrt{\log^2(B/\delta)/D}\right] < \delta.$$

In particular, choosing $\delta = 1/D$ implies that for some constant $C_1 > 0$,

$$\left| \mathop{\mathbb{E}}_{W_B} [\|W_B u\|] - \sqrt{\frac{B}{D}} \right| \leq C_1 \sqrt{\log^2(B/D)/D}.$$

Proof of Theorem F.3. As in the proof of Theorem F.1, we restrict our attention to the case $\|v\| = 1$ and let $w = w_1, \ldots, w_\Delta = Wv$. We note that matrix S being a random uniform permutation implies that every B-block of coordinates in Wv corresponds to picking B coordinates of HT randomly and uniformly without replacement. Therefore, we can apply Lemma F.4 to obtain that for every $i \in [\Delta]$:

$$\left| \mathop{\mathbb{E}}_{W} [\|w_i\|_2] - \sqrt{B/D} \right| = O\left(\frac{\log(B/D)}{\sqrt{D}} \right).$$

In addition, by the permutation symmetry of the distribution of S, we have that

$$\mathop{\mathbb{E}}_{W} [\|w_i\|^2] = \frac{B}{D}.$$

Now, following the same steps as in the proof of Theorem F.1, we have

$$\begin{aligned} \mathop{\mathbb{E}}_{W} \left[\left\| W^\top \mathtt{blkclip}_B(Wv) - v \right\|_2^2 \right] &\leq 2\sqrt{B/D} \cdot \sum_{i \in [\Delta]} \mathop{\mathbb{E}}_{W} \left[\left(\sqrt{B/D} - \|w_i\|_2 \right) \right] \\ &= O\left(\frac{D}{B} \cdot \sqrt{B/D} \cdot \frac{\log(B/D)}{\sqrt{D}} \right) = O\left(\frac{\log(B/D)}{\sqrt{B}} \right). \end{aligned}$$

□

G Communicating 1-sparse vectors

A common application of secure aggregation systems is to aggregate vectors that are 1-sparse (often known as 1-hot vectors) or k-sparse for a small k. Directly using DPFs for these vectors requires $O(D)$ PRG re-seedings and thus can be expensive. Noting that k-sparse vector is also k-block sparse, one can directly use PREAMBLE to reduce the server computation cost at a modest increase in communication cost. Alternately, in settings where we want to add noise in a distributed setting, RAPPOR [EPK14] and its lower-communication variants such as PI-RAPPOR [FT21] and ProjectiveGeometryResponse [FNNT22] can be used along with Prio. Since 1-hot vectors become vectors in $\left\{ \frac{1}{\sqrt{D}}, \frac{-1}{\sqrt{D}} \right\}^D$ after a Hadamard transform, one can view these vectors as Euclidean vectors in $\mathbb{R}^D$ and use PREAMBLE to efficiently communicate them. ProjectiveGeometryResponse is particularly well-suited for this setup even without sampling, as the resulting message space is $O(D)$-dimensional, and a linear transformation of the input space. Thus one can aggregate in "message space": each message is a 1-hot vector in message space, and we can add up these vectors using PREAMBLE. The linear transform to go back to data space is a simple post-processing and the privacy guarantee here can use privacy amplification by shuffling. Since we avoid sampling in this approach, it can scale to larger n without incurring any additional utility overhead.

H k-block-sparse-DPF Construction Details

H.1 Secret-Sharing k-block-sparse Vectors

We first review the tree-based DPF construction from [BGI16], both for single bit outputs and for group element outputs (using our notation). We then describe how we build on their techniques to construct efficient k-block-sparse DPFs.

Tree-based DPF of [BGI16]. The original tree-based DPF construction is formulated for one-sparse vectors without blocks. Let us suppose that the client wants to send a 1-sparse function $f : \{0,1\}^d \to \{0,1\}$ with an input domain size of $D = 2^d$, where the output is non zero only on input $\alpha \in \{0,1\}^d$. Let us define $f_i : \{0,1\}^i \to \{0,1\}$ as the function that computes the sum

$f_i(x) = \sum_{y \in \{0,1\}^{d-i}} f(x||y)$ where $||$ denotes concatenation. Note that $f_d = f$ and each f_i is 1-sparse. Let α_i be the input that produces a non-zero output for f_i.

In the tree-based construction an invariant holds at every layer i in the tree. Servers 1 and 2 hold functions $s_i, t_i : \{0,1\}^i \to \{0,1\}^\lambda$ whose vectors of outputs are secret shares of $r_i \cdot e_{\alpha_i}$, where r_i is a (pseudo) random λ-bit string and e_{α_i} is the basis vector with value 1 at position α_i and 0 elsewhere. In other words, $s_i(x) - t_i(x)$ is zero for $x \neq \alpha_i$ and is a pseudorandom value r_i for $x = \alpha_i$. The servers also hold functions $u_i, v_i : \{0,1\}^i \to \{0,1\}$, whose vectors of outputs are secret shares of e_i. The client knows all secret-shared values.

When $i = 0$, defining these functions is simple: a client with input x sets s_0, t_0 to return a constant λ-bit string chosen at random, and sets $r_0 = s_0(x) - t_0(x)$. It also sets u_0 to return a random bit and $v_0(x) = 1 - u_0(x)$.

For the inductive step, we do the following:

- Each server provisionally expands out their seeds using a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$. We can think of the first half of the PRG's output as the left and the second half as the right child in the tree. They parse for $x \in \{0,1\}^i$:

$$s'_{i+1}(x||0)||u'_{i+1}(x||0)||s'_{i+1}(x||1)||u'_{i+1}(x||1) = G(s_i(x))$$
$$t'_{i+1}(x||0)||v'_{i+1}(x||0)||t'_{i+1}(x||1)||v'_{i+1}(x||1) = G(t_i(x))$$

- Let $\bar{b}_i$ be such that $\alpha_{i+1} \neq \alpha_i||\bar{b}_i$, so that this is the term that needs to be corrected to zero. The client computes the correction according to:

$$c_{i+1} = s'_{i+1}(\alpha_i||\bar{b}_i) - t'_{i+1}(\alpha_i||\bar{b}_i)$$

  and sends it to both servers. The servers then set, for each $x \in \{0,1\}^i$, and $b \in \{0,1\}$:

$$s_{i+1}(x||b) = s'_{i+1}(x||b) - u_i(x)c_{i+1},$$
$$t_{i+1}(x||b) = t'_{i+1}(x||b) - v_i(x)c_{i+1}.$$

  It is then easy to check that for $x \neq \alpha_i$, the equality $u_i(x) = v_i(x)$ implies that $s_{i+1}(x||b) = t_{i+1}(x||b)$. Moreover for $y = \alpha_i||\bar{b}_i$, we have

$$\begin{aligned} s_{i+1}(y) - t_{i+1}(y) &= s'_{i+1}(y) - u_i(\alpha_i)c_{i+1} - t'_{i+1}(y) + v_i(\alpha_i)c_{i+1} \\ &= s'_{i+1}(y) - t'_{i+1}(y) - (u_i(\alpha_i) - v_i(\alpha_i))c_{i+1} \\ &= s'_{i+1}(y) - t'_{i+1}(y) - 1 \cdot c_{i+1} \\ &= 0. \end{aligned}$$

  Here the last step follows by definition of c_{i+1}.

- Finally, we need to correct the bit components. For this purpose, we compute two bit corrections. Recall that $u_{i+1}(y) = v_{i+1}(y)$ for each $y \neq \alpha_i||b$ for $b \in \{0,1\}$. We compute $m_{i+1}(0) = u_{i+1}(\alpha_i||0) - v_{i+1}(\alpha_i||0) + \bar{b}_i$ and similarly $m_{i+1}(1) = u_{i+1}(\alpha_i||1) - v_{i+1}(\alpha_i||1) + (1 - \bar{b}_i)$, and send both of these to the two servers. Similarly to above, the servers now do, for each $x \in \{0,1\}^i$ and each $b \in \{0,1\}$:

$$u_{i+1}(x||b) = u'_{i+1}(x||b) - u_i(x)m_{i+1}(b)$$
$$v_{i+1}(x||b) = v'_{i+1}(x||b) - v_i(x)m_{i+1}(b)$$

  The correctness proof is identical to the one for the $s - t$ case.

Note that the multiplications above all have one of the arguments being bits so this is point-wise multiplication. We have now verified that the invariant holds for $(i + 1)$.

We refer to the tuple $(c_i, m_i(0), m_i(1))$ as a correction word, where c_i is the seed correction and $m_i(0), m_i(1)$ are the correction bits. Intuitively, since $u_i(y) = v_i(y)$ for each $y \neq \alpha_i$, each correction word is only applied to one expanded seed in each level. For all other expanded seeds, the correction word is either never applied (if $u_i(y) = v_i(y) = 0$) or applied twice (if $u_i(y) = v_i(y) = 1$), in which case these two applications cancel out.

Non-zero output from group $\mathbb{G}$. [BGI16] define a variant of their construction where the output of the DPF on input α outputs not 1, but rather a group element $\beta \in \mathbb{G}$. Given a $convert(\cdot)$ function, which converts a λ-bit string to a group element in $\mathbb{G}$, the changes to the construction are minimal. Since the invariant holds, $s_n(\alpha) - t_n(\alpha) \neq 0$ and $s_n(y) - t_n(y) = 0$ for all $y \neq \alpha$. Since the DPF according to Definition B.2 should output β on input α, an additional correction word c_{d+1}, which consists only of a seed correction, is constructed such that either $convert(s_d(\alpha)) - convert(t_d(\alpha)) + c_{d+1} = \beta$ or $convert(s_d(\alpha)) - convert(t_d(\alpha)) - c_{d+1} = \beta$, depending on whether $u_d(\alpha)$ or $v_d(\alpha)$ is one.

The block-sparse case with block size $B > 2$. We adapt the DPF construction of [BGI16] from point functions to block-sparse functions, where the output on any number of input values from a single block will be a non-zero group element. More formally, a block-sparse function $f_{\alpha,\beta}$ with block size B for $\alpha \in \{0,1\}^d$ and $\beta = \{\beta_0, \dots, \beta_{B-1}\} \in \mathbb{G}^B$ is defined to be the function $f : \{0,1\}^{d+\log B} \to \mathbb{G}$ such that $f(\alpha||j) = \beta_j$ and $f(x) = 0$ for $x \neq \alpha||j$ with $j \in [B]$.

To formulate a block-sparse DPF based on tree-based DPF construction, we can use a different PRG for the final tree layer compared to the previous tree layers, such that the output is $B \log |\mathbb{G}|$ bits rather than $2(\lambda + 1)$ bits in the original construction. We call this G'.

$$s'_{d+1}(x||0)|| \dots ||s'_{d+1}(x||B-1) = G'(s_d(x))$$
$$t'_{d+1}(x||0)|| \dots ||t'_{d+1}(x||B-1) = G'(t_d(x))$$

The correction word can be constructed analogously to the original construction, and we make use of a $convert(\cdot)$ function, which maps a $\log |\mathbb{G}|$-length bit string to an element in $\mathbb{G}$. For any input $x = \alpha||j$ for any $j \in [B]$, we set $c_{d+1,j}$ such that $convert(s'_{d+1}(x)) - convert(t'_{d+1}(x)) + c_{d+1,j} = \beta_j$ or $convert(s'_{d+1}(x)) - convert(t'_{d+1}(x)) - c_{d+1,j} = \beta_j$, depending on whether $u_d(\alpha)$ or $v_d(\alpha)$ is one.

This construction avoids the high cost of using the original DPF construction B times, both in terms of computation and communication. In particular, the original construction would involve B DPF keys of size $O(\lambda(d + \log B))$ each, while this construction yields a single DPF key of size $O(\lambda d + B)$. DPF key generation with the original construction will involve $O(d + \log B)$ PRG evaluations for each of the B keys, while our optimization involves $O(d)$ PRG evaluations, as well as one larger PRG evaluation, where the size of this PRG output scales with B. In the evaluation step, when the entire tree is evaluated, naively using the original DPF construction B times would result in $B \cdot 2^{d+\log B}$ PRG evaluations, while our optimization involves only $O(2^d)$ smaller and $O(2^d)$ larger PRG evaluations.

The k-block-sparse case We now describe the idea of a construction for k-block-sparse DPFs, as specified in Definition B.3. Instead of a single index α_i at each level, we have a set $\{\alpha_i^0, \dots, \alpha_i^{\tilde{k}-1}\}$, where $\tilde{k}$ corresponds to the number of distinct i-bit prefixes in α, at most k. We will begin by formulating a change to the construction that allows us to share k-block-sparse vectors, which is a naive extension to the block-sparse version of the DPF construction of [BGI16]. Later, we introduce an optimization to reduce the number of correction words applied to each node.

The invariant on all tree layers except the lowest one is essentially the same as before, except that there are now k separate u and v functions per correction word at each layer, and the client will send k correction words per layer. The PRG at the upper level now outputs $2(k-1)$ additional bits, and the bit components of the expanded and corrected seeds are secret shares of an indicator, which specifies which correction word, if any, will be applied next. As in the original construction, we maintain the goal that each correction word is applied to only at most one expanded seed in that layer. In particular, the correction word with index $\ell \in [\tilde{k}]$ at level $i \in [d]$ will be applied to the expanded seed at position $\alpha_i^\ell \in [2^i]$. In the lowest layer, we can formulate a correction word, which is interpreted as a group element, by applying an idea analogous to that of the block-sparse case.

For the goal of generating DPF keys for a k-block-sparse vector, this construction avoids the overhead of generating kB DPF keys from the original construction. We inherit all advantages of using blocks of size B from the block-sparse construction and obtain further savings by allowing k non-zero blocks. We can compare the the costs of naively using k instantiations of a block-sparse DPF to those of a single k-block-sparse DPF instantiation. Asymptotically, the two approaches require the same amount of communication, with identical DPF key sizes, and the DPF key generation requires the same number of PRG evaluations. The savings for the construction come in the form of computation savings for server evaluation, where the number of PRG evaluations decreases by a factor of k, since servers must now evaluate a single tree, rather than k trees when instantiating a 1-block-sparse DPF k times. However, since each server applies up to k correction words at each level, the total server computation still depends linearly on k. This yields the following result

Theorem H.1. *Let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+2)}$ and $G' : \{0,1\}^\lambda \to \{0,1\}^{B\lceil \log|\mathbb{G}|\rceil}$ be pseudo-random generators. Then there is a scheme $(Gen, Eval)$ that defines a k-block-sparse DPF for the family of k-block-sparse functions $f'_{\alpha,\beta} : \{0,1\}^{d+\log B} \to \mathbb{G}$ with correctness error 0. The key size is $kd(\lambda+4) + kB\lceil \log|\mathbb{G}|\rceil$. In Gen the number of invocations of G is at most kd, and the number of invocations of G' is at most k. In $Eval$ the number of invocations of G is at most d, and G' is invoked once. Evaluating the full vector requires 2^d invocations of G and 2^d invocations of G', and $O(kD)$ additional operations.*

$O(2^{d+\log B})$ PRG calls for each of the k DPF keys, while using our k-block-sparse DPF key generation requires $O(2^d)$ small and $O()$ larger PRG calls.

Cuckoo Hashing. We next show how we can reduce the k-fold multiplicative overhead in the servers' computation using cuckoo hashing. The idea is to only have w control bits (rather than) k for each node in the tree as follows. In practice, we can set the constant w to be between 2 and 5.

At every layer i in the tree, there are k non-zero nodes, which have indices $\{\alpha_i^\ell\}_{\ell\in[k]}$. We use $\tilde{k}$ correction words per layer. Each tree node in the i-th layer is assigned w correction words. These correction words are selected using w hash functions (per layer i), where each hash function maps the 2^i tree nodes to the set of $\tilde{k}$ correction words. The hash functions are public and known to both servers (they are chosen independently of the values of the DPF). The client assigns each non-zero node to one of the w correction words specified (for that node) by the hash functions. This assignment does depend on the non-zero indices of the DPF and must not be known to the servers. Cuckoo hashing [PR01] shows that for any set of k non-zero nodes (specified by the values $\{\alpha_i^\ell\}$), except with probability $\tilde{\mathcal{O}}(\frac{1}{k})$ over the choice of the hash functions, the client can choose the assignment so that there are no "collisions": no two non-zero nodes are mapped to the same correction word. In the case of failure, which occurs with probability at most $\tilde{\mathcal{O}}(\frac{1}{k})$, the client will generate a outputs keys corresponding to the zero vector, which can trivially be realized by picking an arbitrary assignment. This does not affect the security of the construction as the failure of cuckoo hashing is not revealed. It does however mean that the correct vector is sent with probability $1 - O(\frac{1}{k})$, rather than 1. For statistical applications, this small failure probability has little impact.

The correction words are now constructed and applied as usual, except for the fact that each correction word now has only w correction bits instead of k on each side and that one of only w correction words is applied per expanded seed/node. The bit components of an expanded and corrected seed still correspond to an indicator specifying which single correction word is applied at the next layer, as before; however, it is no longer up to one of all k possible correction words that will be applied, but rather one of the w possible correction words specified by the w hash functions.

For simplicity, we present the formal construction by setting $w = 2$. The details can be found in Figures 7 and 9, using a helper function for DPF key generation in Figure 8 to specify the constructions at each of the upper tree layers. Note that we formulate evaluation in Figure 9 for a single path in the tree for simplicity; to reconstruct the entire vector instead of just one entry, all nodes in the tree can be evaluated using the same approach. In that case, the number of invocations of G is at most 2^d, and the number of invocations of G' is at most 2^d.

Theorem H.2. *Let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+2)}$ and $G' : \{0,1\}^\lambda \to \{0,1\}^{B\lceil \log|\mathbb{G}|\rceil}$ be pseudo-random generators. Also suppose $\{h_i\}_{i\in[d+1]} : [2^{i-1}] \to [ck]^2$ describes a set of random hash functions. Then the scheme $(Gen, Eval)$ from Figures 7 and 9 is a k-block-sparse DPF for the family of k-block-sparse functions $f'_{\alpha,\beta} : \{0,1\}^{d+\log B} \to \mathbb{G}$ with correctness error $\tilde{O}(\frac{1}{k})$. The key size is $3kd(\lambda+4) + 3kB\lceil \log|\mathbb{G}|\rceil$. In Gen the number of invocations of G is at most kd, and the number of invocations of G' is at most k. In $Eval$ the number of invocations of G is at most d, and G' is invoked once. Evaluating the full vector requires 2^d invocations of G and 2^d invocations of G', and $O(D)$ additional operations.*

Remark H.3. *Note that cuckoo hashing can yield different trade-offs from those in Theorem H.2 if more than 2 hash functions are used. For $w = 2$, the total number of required correction words to achieve a low failure probability is approximately $3k$. If $w = 4$, the total number of required correction words to achieve a low failure probability can be reduced to be only about $1.03k$ [FP12, FM12], leading to a key size of $1.03kd(\lambda + 4) + 1.03kB\lceil \log |\mathbb{G}| \rceil$. Increasing w decreases the total number of correction words, and therefore the total key size and required communication, by a constant factor, at the cost of increasing the total number of field operations per node by a constant factor. The number of PRG evaluations does not depend on w.*

Remark H.4. *Recall from related work the application of cuckoo hashing to multi-point function secret sharing [ACLS18, DRRT18, SGRR19, dCP22], where cuckoo hashing is applied directly to the k-sparse 2^d-dimensional secret vector, constructing k smaller vectors with a single entry each before generating DPF keys. In this approach, the total number of vector entries, and therefore also the number of PRG evaluations, is $2 \cdot 2^d$ or $3 \cdot 2^d$ when 2 or 3 hash functions are used for cuckoo hashing, as suggested by [DRRT18]. Because our application of cuckoo hashing is at the level of correction words within the DPF construction, it avoids this overhead and requires only 2^d vector entries.*

Proof of Theorem H.2. We prove both correctness and security of the scheme.

Correctness. We describe and argue correctness of our optimized k-block-sparse DPF construction in a way that is analogous to our arguments for the original construction of [BGI16]. The invariant for the k-sparse case is that in layer i of the tree construction, the $\tilde{k}$ nodes corresponding to α_i are non-zero, and all others are zero. In addition, we maintain the invariant that exactly one of the two bit components on the non-zero path is 1, and the other is 0. More formally, we would like that if $x \notin \alpha_i$, $s_i(x) = t_i(x)$, $u_i(x) = v_i(x)$, and $q_i(x) = r_i(x)$. Furthermore, we would like that for $x \in \alpha_i$, exactly one of $u_i(x) = v_i(x)$, and $q_i(x) = r_i(x)$ should hold.

The function $h_i : [2^{i-1}] \to [3k]^2$ maps one α_i^ℓ in tree layer i to two correction words. We use cuckoo hashing to determine which of these two correction words will be applied for α_i^ℓ, defining function $g_i : [2^{i-1}] \to \{0, 1\}$. Due to cuckoo hashing, we know that this mapping exists for any k-block-sparse function given all h_i with probability $1 - \mathcal{O}(\frac{1}{k})$. In the upper levels, the PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+4}$ output is parsed as follows:

$$
\begin{aligned}
s'_{i+1}(x||0)||u'_{i+1}(x||0)||q'_{i+1}(x||0)||s'_{i+1}(x||1)||u'_{i+1}(x||1)||q'_{i+1}(x||1) = G(s_i(x)) \\
t'_{i+1}(x||0)||v'_{i+1}(x||0)||r'_{i+1}(x||0)||t'_{i+1}(x||1)||v'_{i+1}(x||1)||r'_{i+1}(x||1) = G(t_i(x))
\end{aligned}
$$

In the original construction, the seed portion of the correction word was set in such a way as to set to zero the node corresponding to $\alpha_i||\bar{b}_i$, where $\bar{b}_i$ is defined such that $\alpha_{i+1} \neq \alpha_i||\bar{b}_i$. Let us now analogously define $\bar{b}_i^\ell$, defined such that $\alpha_{i+1}^\ell \neq \alpha_i^\ell||\bar{b}_i^\ell$. It is possible that there exist indices $\ell \neq \ell' \in [k]$ such that $\alpha_i^\ell||\bar{b}_i^\ell = \alpha_{i+1}^{\ell'}$, in which case we do not want to set the seed portion of the node corresponding to $\alpha_i^\ell||\bar{b}_i^\ell$ to 0. For such an ℓ, we set the seed portion of the corresponding correction word to a random bit-string instead. For other ℓ, we set the corresponding seed correction, specified by the $g_i(\alpha_i^\ell)$th output of $h_i(\alpha_i^\ell)$, as expected:

$$
c_{h_{i+1}(\alpha_i^\ell)[g_i(\alpha_i^\ell)]} = s'_{i+1}(\alpha_i^\ell||\bar{b}_i^\ell) + t'_{i+1}(\alpha_i^\ell||\bar{b}_i^\ell)
$$

The corrected seed components are then for each $x \in \{0,1\}^i$ and $b \in \{0,1\}$:

$$
\begin{aligned}
s_{i+1}(x||b) = s'_{i+1}(x||b) - u_i(x)c_{h_{i+1}(x)[0]} - q_i(x)c_{h_{i+1}(x)[1]} \\
t_{i+1}(x||b) = t'_{i+1}(x||b) - v_i(x)c_{h_{i+1}(x)[0]} - r_i(x)c_{h_{i+1}(x)[1]}
\end{aligned}
$$

It is then easy to check that for $x \neq \alpha_i^\ell$ for all $\ell \in [k]$, the equalities $u_i(x) = v_i(x)$ and $q_i(x) = r_i(x)$ imply that $s_{i+1}(x||b) = t_{i+1}(x||b)$. Moreover for $y = \alpha_i^\ell || \bar{b}_i^\ell$, as long as $y \notin \alpha_{i+1}$, we have

$$\begin{aligned}
s_{i+1}(y) - t_{i+1}(y) &= s'_{i+1}(y) - u_i(\alpha_i^\ell) c_{h_{i+1}(\alpha_i^\ell)[0]} - q_i(\alpha_i^\ell) c_{h_{i+1}(\alpha_i^\ell)[1]} \\
&\quad - (t'_{i+1}(y) - v_i(\alpha_i^\ell) c_{h_{i+1}(\alpha_i^\ell)[0]} - r_i(\alpha_i^\ell) c_{h_{i+1}(\alpha_i^\ell)[1]}) \\
&= s'_{i+1}(y) - t'_{i+1}(y) - (u_i(\alpha_i^\ell) - v_i(\alpha_i^\ell)) c_{h_{i+1}(\alpha_i^\ell)[0]} \\
&\quad - (q_i(\alpha_i^\ell) - r_i(\alpha_i^\ell)) c_{h_{i+1}(\alpha_i^\ell)[1]} \\
&= s'_{i+1}(y) - t'_{i+1}(y) - 1 c_{h_{i+1}(\alpha_i^\ell)[g_{i+1}(\alpha_i^\ell)]} \\
&= 0.
\end{aligned}$$

Here the last step follows by definition of $c_{h_{i+1}(\alpha_i^\ell)[g_{i+1}(\alpha_i^\ell)]}$.

Finally, we need to correct the new bit components. For this purpose, we compute two bit corrections. Note that $u_{i+1}(y) = v_{i+1}(y)$ and $q_{i+1}(y) = r_{i+1}(y)$ for each $y \notin \alpha_{i+1}$. On the other hand, when $y \in \alpha_{i+1}$, we would like either $u_{i+1}(y) = v_{i+1}(y)$ and $q_{i+1}(y) \neq r_{i+1}(y)$ or $u_{i+1}(y) \neq v_{i+1}(y)$ and $q_{i+1}(y) = r_{i+1}(y)$, depending on $g_{i+1}(y)$. We set the bit corrections to:

$$\begin{aligned}
m_{h_{i+1}(\alpha_i^\ell)[g_{i+1}(\alpha_i^\ell)]}(0) &= u'_{i+1}(\alpha_i^\ell||0) - v'_{i+1}(\alpha_i^\ell||0) + (g_{i+2}(\alpha_i^\ell||0) + 1)(\alpha_i^\ell||0 \in \alpha_{i+1}) \\
m_{h_{i+1}(\alpha_i^\ell)[g_{i+1}(\alpha_i^\ell)]}(1) &= u'_{i+1}(\alpha_i^\ell||1) - v'_{i+1}(\alpha_i^\ell||1) + (g_{i+2}(\alpha_i^\ell||1) + 1)(\alpha_i^\ell||1 \in \alpha_{i+1}) \\
p_{h_{i+1}(\alpha_i^\ell)[g_{i+1}(\alpha_i^\ell)]}(0) &= q'_{i+1}(\alpha_i^\ell||0) - r'_{i+1}(\alpha_i^\ell||0) + g_{i+2}(\alpha_i^\ell||0)(\alpha_i^\ell||0 \in \alpha_{i+1}) \\
p_{h_{i+1}(\alpha_i^\ell)[g_{i+1}(\alpha_i^\ell)]}(1) &= q'_{i+1}(\alpha_i^\ell||1) - r'_{i+1}(\alpha_i^\ell||1) + g_{i+2}(\alpha_i^\ell||1)(\alpha_i^\ell||1 \in \alpha_{i+1})
\end{aligned}$$

To apply the bit correction, the following is computed:

$$\begin{aligned}
u_{i+1}(x||b) &= u'_{i+1}(x||b) - u_i(x) m_{h_{i+1}(x)[0]}(b) - q_i(x) m_{h_{i+1}(x)[1]}(b) \\
v_{i+1}(x||b) &= v'_{i+1}(x||b) - v_i(x) m_{h_{i+1}(x)[0]}(b) - r_i(x) m_{h_{i+1}(x)[1]}(b) \\
q_{i+1}(x||b) &= q'_{i+1}(x||b) - u_i(x) p_{h_{i+1}(x)[0]}(b) - q_i(x) p_{h_{i+1}(x)[1]}(b) \\
r_{i+1}(x||b) &= r'_{i+1}(x||b) - v_i(x) p_{h_{i+1}(x)[0]}(b) - r_i(x) p_{h_{i+1}(x)[1]}(b)
\end{aligned}$$

The correctness proof is identical to the one for the $s - t$ case when $x||b \notin \alpha_{i+1}$. Otherwise, when $x||b \in \alpha_{i+1}$, we show that

$$\begin{aligned}
u_{i+1}(x||b) - v_{i+1}(x||b) &= u'_{i+1}(x||b) - u_i(x) m_{h_{i+1}(x)[0]}(b) - q_i(x) m_{h_{i+1}(x)[1]}(b) \\
&\quad - (v'_{i+1}(x||b) - v_i(x) m_{h_{i+1}(x)[0]}(b) - r_i(x) m_{h_{i+1}(x)[1]}(b)) \\
&= u'_{i+1}(x||b) - v'_{i+1}(x||b) - (u_i(x) - v_i(x)) m_{h_{i+1}(x)[0]}(b) \\
&\quad - (q_i(x) r_i(x)) m_{h_{i+1}(x)[1]}(b) \\
&= u'_{i+1}(x||b) - v'_{i+1}(x||b) - m_{h_{i+1}(x)[g_{i+1}(x)]}(b) \\
&= g_{i+2}(\alpha_i^\ell||b) + 1
\end{aligned}$$

Using analogous arguments, $q_{i+1}(x||b) - r_{i+1}(x||b) = g_{i+2}(\alpha_i^\ell||b)$. Therefore, exactly one of either $u_{i+1}(x||b) = v_{i+1}(x||b)$ or $q_{i+1}(y) = r_{i+1}(y)$ can hold, and the required invariant is maintained.

In the lowest layer, the PRG is evaluated for each $\ell \in [k]$:

$$\begin{aligned}
s'_{d+1}(\alpha^\ell||0)|| \ldots ||s'_{d+1}(\alpha^\ell||B-1) &= G'(s_d(\alpha^\ell)) \\
t'_{d+1}(\alpha^\ell||0)|| \ldots ||t'_{d+1}(\alpha^\ell||B-1) &= G'(t_d(\alpha^\ell))
\end{aligned}$$

In a next step, we call the $convert(\cdot)$ function on each component of these outputs. The goal is to formulate one correction word for each of the k non-zero blocks.

More formally, we would like to choose correction words $c_{h_{d+1}(\alpha^\ell)[g_{d+1}(\alpha^\ell)]}(j)$ for $j \in [B]$ such that $s_{d+1}(\alpha^\ell||j) - t_{d+1}(\alpha^\ell||j) = \beta_j^\ell$:

$$\begin{aligned}
s_{d+1}(\alpha^\ell||j) &= s'_{d+1}(\alpha^\ell||j) + u_d(\alpha^\ell) c_{h_{d+1}(\alpha^\ell)[0]}(j) + q_d(\alpha^\ell) c_{h_{d+1}(\alpha^\ell)[1]}(j) \\
t_{d+1}(\alpha^\ell||j) &= t'_{d+1}(\alpha^\ell||j) + v_d(\alpha^\ell) c_{h_{d+1}(\alpha^\ell)[1]}(j) + r_d(\alpha^\ell) c_{h_{d+1}(\alpha^\ell)[1]}(j).
\end{aligned}$$

Note that for $x \notin \alpha$, $s'_{d+1}(x||j) = t'_{d+1}(x||j)$, $u_d(x) = v_d(x)$, and $q_d(x) = r_d(x)$, so $s_{d+1}(x||j) = t_{d+1}(x||j)$ for all $j \in [B]$, regardless of the correction words.

For α^ℓ, if $g_{d+1}(\alpha^\ell) = 0$, then $u_d(\alpha^\ell) \neq v_d(\alpha^\ell)$ and $q_d(\alpha^\ell) = r_d(\alpha^\ell)$. Otherwise, $q_d(\alpha^\ell) \neq r_d(\alpha^\ell)$ and $u_d(\alpha^\ell) = v_d(\alpha^\ell)$. Exactly one of $s_{d+1}(x||j)$ and $t_{d+1}(x||j)$ is independent of $c_{h_{d+1}(\alpha^\ell)[g_{d+1}(\alpha^\ell)]}$. By subtracting over $\mathbb{G}$, we can find the target value of the other one and choose the correction word accordingly.

In a bit more detail: the client computes the sequence $(s'_{d+1}(\alpha^\ell||j) - t'_{d+1}(\alpha^\ell||j))_{j\in[B]} \in \mathbb{G}$. It computes the seed correction for each $j \in [B]$ and send it to both servers.. If $g_{d+1}(\alpha^\ell) = 0$

$$c_{h_{d+1}(\alpha^\ell)[g_{d+1}(\alpha^\ell)]}(j) = (u_{d,\ell}(\alpha^\ell) - v_{d,\ell}(\alpha^\ell))^{-1}(\beta_j^\ell - (s'_{d+1}(\alpha^\ell||j) - t'_{d+1}(\alpha^\ell||j)))$$

Otherwise:

$$c_{h_{d+1}(\alpha^\ell)[g_{d+1}(\alpha^\ell)]}(j) = (q_d(\alpha^\ell) - r_d(\alpha^\ell))^{-1}(\beta_j^\ell - (s'_{d+1}(\alpha^\ell||j) - t'_{d+1}(\alpha^\ell||j)))$$

Since $(u_d(\alpha^\ell) - v_d(\alpha^\ell)), (q_d(\alpha^\ell) - r_d(\alpha^\ell)) \in \{-1, 0, 1\}$, the inverse over $\mathbb{G}$ is well defined.

Now for $x = \alpha^\ell$, we can write

$$\begin{aligned}
&s_{d+1}(\alpha^\ell||j) - t_{d+1}(\alpha^\ell||j)\\
&\quad = s'_{d+1}(\alpha^\ell||j) - t'_{d+1}(\alpha^\ell||j) + (u_d(\alpha^\ell) - v_d(\alpha^\ell))c_{h_{d+1}(\alpha^\ell)[0]}(j)\\
&\qquad + (q_d(\alpha^\ell) - r_d(\alpha^\ell))c_{h_{d+1}(\alpha^\ell)[1]}(j)\\
&\quad = s'_{d+1}(\alpha^\ell||j) - t'_{d+1}(\alpha^\ell||j) + (\beta_j^\ell - s'_{d+1}(\alpha^\ell||j) + t'_{d+1}(\alpha^\ell||j))\\
&\quad = \beta_j^\ell.
\end{aligned}$$

Security. We argue that each server's DPF key is pseudorandom. The security proof is analogous to the proof of Theorem 3.3 in [BGI16]. We begin by describing the high-level argument. Each server begins with a random seed, which is unknown to the other server. In each tree layer, up to k random seeds are expanded using a PRG, generating 2 new seeds and 4 bits, all of which appear similarly random due to the security of the PRG and the fact that the original seed appeared random. The application of a correction word will cancel the randomness of 5 of these 6 resulting seed and bit components. Specifically, one seed component and all 4 bit components are canceled; however, given only the correction word and the tree node values held by a single server, the resulting secret shares still appear random.

It is possible to define a series of hybrids $Hyb_{w,\ell}$, where the correction words in all levels $i < w$ are replaced by random bit strings for $i \in [d+1]$, replacing Step 2 in Figure 8, the first ℓ correction word components in layer w are replaced by a random bit string, and if $w = d+1$ the first ℓ components of the final level correction word are replaced with random group elements, replacing Step 9 in Figure 7. We also consider the view of the first of the two servers, without loss of generality. Specifically:

1. Choose $s_0(0), t_0(0), u_0(0), v_0(0), q_0(0), r_0(0)$ honestly.
2. Choose $CW^0, \ldots, CW^{w-1} \in \{0,1\}^{3k(\lambda+4)}$ uniformly at random.
3. Choose CW^w such that the first ℓ components are uniform samples and the remaining ones are computed honestly.
4. Update $s_i(\alpha_i^\ell)$, $u_i(\alpha_i^\ell)$, $q_i(\alpha_i^\ell)$ honestly for all $i < w$ and $\ell \in [\tilde{k}]$.
5. For $i = w$, set $t_i(\alpha_i^0)$,$v_i(\alpha_i^0)$,$r_i(\alpha_i^0)$, $\ldots$, $t_i(\alpha_i^\ell)$,$v_i(\alpha_i^\ell)$,$r_i(\alpha_i^\ell)$, as well as set $t_{i-1}(\alpha_{i-1}^{\ell+1})$,$v_{i-1}(\alpha_{i-1}^{\ell+1})$,$r_{i-1}(\alpha_{i-1}^{\ell+1})$, $\ldots$, $t_{i-1}(\alpha_{i-1}^{\tilde{k}-1})$,$v_{i-1}(\alpha_{i-1}^{\tilde{k}-1})$,$r_{i-1}(\alpha_{i-1}^{\tilde{k}-1})$, to random samples. Compute $t_i(\alpha_i^{\ell+1})$,$v_i(\alpha_i^{\ell+1})$,$r_i(\alpha_i^{\ell+1})$, $\ldots$, $t_i(\alpha_i^{\tilde{k}-1})$,$v_i(\alpha_i^{\tilde{k}-1})$,$r_i(\alpha_i^{\tilde{k}-1})$ honestly.
6. For $i > w$, compute all CW^i and update all $s_i(\alpha_i^\ell)$, $t_i(\alpha_i^\ell)$, $u_i(\alpha_i^\ell)$, $v_i(\alpha_i^\ell)$, $q_i(\alpha_i^\ell)$, $r_i(\alpha_i^\ell)$ honestly for all $\ell \in [k]$.
7. The output is $s_0(0), u_0(0), q_0(0)||CW^1||\ldots||CW^{d+1}$.

Note that when $w = \ell = 0$, this experiment corresponds to the honest key distribution, whereas when $w = d+1, \ell = k-1$ this yields a completely random key. We claim that each pair of adjacent hybrids will be indistinguishable based on the security of the pseudorandom generator.

We first consider $w \leq d$ and a Hyb-distinguishing adversary $\mathcal{A}$ who distinguishes $Hyb_{w,\ell}$ from either $Hyb_{w,\ell+1}$ if $\ell < \tilde{k} - 1$ or $Hyb_{w+1,0}$ otherwise. Given an adversary $\mathcal{A}$ with advantage ρ, we can construct a corresponding PRG adversary $\mathcal{B}$. This PRG adversary is given a value $r \in \{0,1\}^{2(\lambda+2)}$ and distinguishes between the cases where r is truly random and $r = G(s)$, where $s \in \{0,1\}^\lambda$ is a random seed. Given α, β, w, ℓ, the adversary $\mathcal{B}$ constructs a DPF key according to $Hyb_{w,\ell}$; however, instead of sampling $t_w(\alpha^\ell_w), v_w(\alpha^\ell_w), r_w(\alpha^\ell_w)$ randomly, we set:

$$t_w(\alpha^\ell_{w-1}||0)||v_w(\alpha^\ell_{w-1}||0)||r_w(\alpha^\ell_{w-1}||0)||t_w(\alpha^\ell_{w-1}||1)||v_w(\alpha^\ell_{w-1}||1)||r_w(\alpha^\ell_{w-1}||1)$$

to r. If r is computed pseudorandomly, then it is clear that the resulting DPF key is generated as in $Hyb_{w,\ell}(1^\lambda, \alpha, \beta)$. We must also argue that if r is random, the resulting key is distributed as in either $Hyb_{w,\ell+1}$ if $\ell < \tilde{k} - 1$ or $Hyb_{w+1,0}$ otherwise. If $t_w(\alpha^\ell_w), v_w(\alpha^\ell_w), r_w(\alpha^\ell_w)$ is random, then the corresponding correction word is also uniformly random, since it is computed as the xor of a fixed bit-string with these randomly selected bit-strings, forming a perfect one-time pad. After applying this correction word, the resulting seed and bit components are also uniformly distributed, given only the previous seed and bit components for that server, as well as the correction words.

Combining these pieces, an adversary $\mathcal{A}$ that distinguishes between the hybrids with advantage ρ yields a corresponding adversary $\mathcal{B}$ for the PRG experiment with the same advantage and only polynomial additional runtime.

Finally, we consider $w = d + 1$. We can make an argument similar to the previous one that an adversary that distinguishes between the distributions $Hyb_{d+1,\ell}$ and $Hyb_{d+1,\ell+1}$ with advantage ρ directly yields a corresponding adversary $\mathcal{B}$ for the pseudo-randomness of the PRG output, interpreted as a group element, with the same advantage and only polynomial additional runtime. $\mathcal{B}$ can embed the challenge by setting the corresponding correction word to $(-1)^{r_{i-1}(\alpha^\ell_{i-1})}(\beta^\ell_j - s'_i(\alpha^\ell_{i-1}||j) + r)$. If r is generated pseudo-randomly, this is exactly the distribution of $Hyb_{d+1,\ell}$. If r is truly random, then it similarly acts as a one-time pad on the remaining terms and the corresponding correction word is uniformly distributed, as in $Hyb_{d+1,\ell+1}$. $\square$

H.2 Proofs of Validity

We construct an efficient proof-system that allows a client to prove that it shared a valid block-sparse DPF. The proof is divided into two components:

1. k correction-bit sparse. The client proves that at most k of the (secret shared) correction bits are non-zero.
2. k block-sparse. Given that at most k of the correction bits are non-zero, the client proves that there are at most k non-zero blocks in the output.

We detail these components below. The proof system is sound against a malicious client, but we assume semi-honest behavior by the servers.

Theorem H.5. *The scheme of Theorem H.2 can be augmented to be a verifiable DPF for the same function family (k-block-sparse functions). The construction incurs an additional round of interaction between the client and the servers (this can be eliminated using the Fiat-Shamir heuristic). The soundness error is $(poly\,(k)/2^\lambda)$.*

The additional cost for the proof (on top of the construction above) is $O(k \cdot d \cdot \lambda)$ communication, $(k \cdot poly\,(d, \lambda))$ client work, $(k \cdot 2^d \cdot poly\,(d, \lambda))$ server work for each server.

Proving k-sparsity of the correction bits. The secret shares for the correction bits are in $\{0,1\}$ (the correction bit is "on" if these bit values are not identical). The client proves that the vector of $2 \cdot 2^d$ correction bits (2^d pairs) is k-sparse, i.e. at most k of the bits are non-zero. We use the efficient construction of 1-sparse DPFs from [BGI16], which comes with an efficient proof system (the construction in [BBCG+21] also handles malicious servers, but we do not treat this case here). The client sends k 1-sparse DPFs (unit vectors over $\{0,1\}^{2^{d+1}}$) whose sum equals the vector of

Gen

$Gen(1^\lambda, \alpha, \beta, \mathbb{G}, B, k)$:

1. Let $\alpha = \{\alpha^\ell\}_{\ell\in[k]} \in \{\{0,1\}^d\}^k$
2. Sample random $s_0(0) \leftarrow \{0,1\}^\lambda$ and $t_0(0) \leftarrow \{0,1\}^\lambda$
3. For i from 1 to $d+1$, use cuckoo hashing to define mapping functions $h_i : [2^{i-1}] \to [ck]^2$ and $g_i : \{(i-1)\text{-bit prefixes in } \alpha\} \to \{0,1\}$
4. If $g_1(0) = 0$, let $u_0(0) = 0$, $v_0(0) = 1$, $q_0(0) = 0$, and $r_0(0) = 0$. Else let $u_0(0) = 0$, $v_0(0) = 0$, $q_0(0) = 0$, and $r_0(0) = 1$.
5. For i from 1 to d:
   - Compute $GenNext(\alpha, i, s_{i-1}, t_{i-1}, u_{i-1}, v_{i-1}, q_{i-1}, r_{i-1}, h_i, g_i, g_{i+1})$ and parse the output as $CW^i, s_i, t_i, u_i, v_i, q_i, r_i$
6. group α by entries with the same d-bit prefix
7. For each distinct (d-bit prefixes in α, denoted α^ℓ:
   - $s'_{d+1}(\alpha^\ell||0)||...||s'_{d+1}(\alpha^\ell||B-1) \leftarrow G'(s_d(\alpha^\ell))$
   - $t'_{d+1}(\alpha^\ell||0)||...||t'_{d+1}(\alpha^\ell||B-1) \leftarrow G'(t_d(\alpha^\ell))$
   - For $j \in [B]$, convert $s'_{d+1}(\alpha^\ell||j) := convert(s'_{d+1}(\alpha^\ell||j))$ and $t'_{d+1}(\alpha^\ell||j) := convert(t'_{d+1}(\alpha^\ell||j))$
8. Parse $\beta = (\beta^0, \ldots, \beta^{k-1})$
9. For $\ell \in [k]$:
   - Denote α^ℓ the d-bit prefix associated with ℓ. Also denote $\rho = h_{d+1}(\alpha^\ell)$.
   - Parse $\beta^\ell = (\beta^\ell_0, \ldots, \beta^\ell_{B-1})$
   - Denote $\gamma^\ell_j = \beta^\ell_j - s'_{d+1}(\alpha^\ell||j) + t'_{d+1}(\alpha^\ell||j)$ for $j \in [B]$.
   - Denote $c_{\rho[g_{d+1}(\alpha^\ell)]}(j) = (-1)^{v_d(\alpha^\ell)} \cdot \gamma^\ell_j$
   - If $g_i(\alpha^\ell) = 0$, set
     $CW^{d+1}_{\rho[0]} \leftarrow c_{\rho[g_i(\alpha^\ell)]}(0)||...||c_{\rho[g_i(\alpha^\ell)]}(B-1)$.
   - Else, set
     $CW^{d+1}_{\rho[1]} \leftarrow (-1)^{r_d(\alpha^\ell)} \cdot \gamma^\ell_0||...||(-1)^{r_d(\alpha^\ell)} \cdot \gamma^\ell_{B-1}$.
10. For remaining ℓ, set CW^{d+1}_ℓ randomly
11. Set $CW^{d+1} = CW^{d+1}_1||...||CW^{d+1}_k$ and $CW = CW^1|| \ldots ||CW^{d+1}||h_1|| \ldots ||h_{d+1}$, as well as $k_0 = s_0(0)||CW$, $k_1 = t_0(0)||CW$.
12. return (k_0, k_1), $g_1(0)$

Figure 7: Gen generates DPF keys for a k-block-sparse vector of dimension $d + \log B$ with blocks of size B and security parameter λ, where G' is a PRG that takes an input of size λ bits and outputs a bit string of length $B \log |\mathbb{G}|$. The values β of the non-zero entries in the vector correspond to elements of group $\mathbb{G}$.

correction bits: if these extra DPFs are indeed one-sparse and sum up to the vector of correction bits, then that vector must be k-sparse. We remark that we are agnostic to the field used for secret-sharing these additional DPFs (the secret shares of the correction bits are treated as the 0 and the 1 element in the field being used). Soundness and zero-knowledge follow from the properties of the DPF of [BGI16]. The communication is $O(k \cdot d \cdot \lambda)$, the client runtime is $(k \cdot \text{poly}(d, \lambda))$, and the server runtime is $(k \cdot 2^d \cdot \text{poly}(d, \lambda))$. The soundness error is $O(k/2^\lambda)$.

Proving k-sparsity of the output blocks. Given that at most k of the correction bits are non-zero, the client needs to prove that there are at most k non-zero blocks in the output. Consider the final layer of the DPF tree: in a zero block, the two PRG seeds held by the servers are identical, whereas in a non-zero block, they are different. Rather than expanding the seeds to B group elements (as in the vanilla construction above), we add another λ bits to the output, and we also add λ corresponding bits to each correction word. We refer to these as the check-bits of the PRG outputs / correction words, and we refer to the original outputs (the B group elements) as the payload bits. In the zero blocks the check-bits of the outputs should be identical: subtracting them should results in a zero vector. In each non-zero blocks, the check-bits of the (appropriate) correction word are chosen so that subtracting them from the (subtraction of the) check-bits of the PRG outputs also results in a zero vector. Thus, in our proof system, the servers verify that, in each block, the appropriate

GenNext

$GenNext(\alpha, i, s_{i-1}, t_{i-1}, u_{i-1}, q_{i-1}, v_{i-1}, r_{i-1}, h_i, g_i, g_{i+1})$:

1. Group α by entries with the same $(i-1)$-bit prefix. For each group:
   - Denote α^{ℓ}_{i-1} the $(i-1)$-bit prefix associated with that group
   - Expand and parse $s'_i(\alpha^{\ell}_{i-1}||0)||u'_i(\alpha^{\ell}_{i-1}||0)||q'_i(\alpha^{\ell}_{i-1}||0)||s'_i(\alpha^{\ell}_{i-1}||1)||$ $u'_i(\alpha^{\ell}_{i-1}||1)||q'_i(\alpha^{\ell}_{i-1}||1) \leftarrow G(s_{i-1}(\alpha^{\ell}_{i-1}))$
   - Expand and parse $t'_i(\alpha^{\ell}_{i-1}||0)||v'_i(\alpha^{\ell}_{i-1}||0)||r'_i(\alpha^{\ell}_{i-1}||0)||$ $t'_i(\alpha^{\ell}_{i-1}||1)||v'_i(\alpha^{\ell}_{i-1}||1)||r'_i(\alpha^{\ell}_{i-1}||1) \leftarrow G(t_{i-1}(\alpha^{\ell}_{i-1}))$
2. For each group of $(i-1)$-bit prefixes:
   - Denote α^{ℓ}_{i-1} the $(i-1)$-bit prefix associated with that group. Also denote $\rho = h_i(\alpha^{\ell}_{i-1})$.
   - If α contains values with prefixes $\alpha^{\ell}_{i-1}||0$ and $\alpha^{\ell}_{i-1}||1$, set $c_{\rho[g_i(\alpha^{\ell}_{i-1})]}$ random
   - Else if α contains values with prefixes $\alpha^{\ell}_{i-1}||0$, set $c_{\rho[g_i(\alpha^{\ell}_{i-1})]} = s'_i(\alpha^{\ell}_{i-1}||1) + t'_i(\alpha^{\ell}_{i-1}||1)$
   - Else if α contains values with prefixes $\alpha^{\ell}_{i-1}||1$, set $c_{\rho[g_i(\alpha^{\ell}_{i-1})]} = s'_i(\alpha^{\ell}_{i-1}||0) + t'_i(\alpha^{\ell}_{i-1}||0)$
   - For $j \in [2]$:
     - If α contains a value with prefix $\alpha^{\ell}_{i-1}||j$, set $m_{\rho[g_i(\alpha^{\ell}_{i-1})]}(j) = u'_i(\alpha^{\ell}_{i-1}||j) + v'_i(\alpha^{\ell}_{i-1}||j) + 1 + g_{i+1}(\alpha^{\ell}_{i-1}||j)$ and $p_{\rho[g_i(\alpha^{\ell}_{i-1})]}(j) = q'_i(\alpha^{\ell}_{i-1}||j) + r'_i(\alpha^{\ell}_{i-1}||j) + g_{i+1}(\alpha^{\ell}_{i-1}||j)$
     - Else, set $m_{\rho[g_i(\alpha^{\ell}_{i-1})]}(j) = u'_i(\alpha^{\ell}_{i-1}||j) + v'_i(\alpha^{\ell}_{i-1}||j)$ and $p_{\rho[g_i(\alpha^{\ell}_{i-1})]}(j) = q'_i(\alpha^{\ell}_{i-1}||j) + r'_i(\alpha^{\ell}_{i-1}||j)$
3. For all indeces ℓ that have not been set yet, set c_ℓ to a new random sample, and set $m_\ell(j)$ and $p_\ell(j)$ to random bits for all $j \in [2]$.
4. Parse $CW^i = c_0||m_0(0)||p_0(0)||m_0(1)||p_0(1)||...$ $||c_{ck-1}||m_{ck-1}(0)||p_{ck-1}(0)||m_{ck-1}(1)||p_{ck-1}(1)$
5. Group α by entries with the same $(i-1)$-bit prefix α^{ℓ}_{i-1}. For each group: For $j \in [2]$: If α contains a value with prefix $\alpha^{\ell}_{i-1}||j$:
   - Denote $\rho = h_i(\alpha^{\ell}_{i-1})$.
   - Set $s_i(\alpha^{\ell}_{i-1}||j) \leftarrow s'_i(\alpha^{\ell}_{i-1}||j) + u_{i-1}(\alpha^{\ell}_{i-1}) \cdot c_{\rho[0]} + q_{i-1}(\alpha^{\ell}_{i-1}) \cdot c_{\rho[1]}$ and $t_i(\alpha^{\ell}_{i-1}||j) \leftarrow t'_i(\alpha^{\ell}_{i-1}||j) + v_{i-1}(\alpha^{\ell}_{i-1}) \cdot c_{\rho[0]} + r_{i-1}(\alpha^{\ell}_{i-1}) \cdot c_{\rho[1]}$
   - Set
     $u_i(\alpha^{\ell}_{i-1}||j) \leftarrow u'_i(\alpha^{\ell}_{i-1}||j) + u_{i-1}(\alpha^{\ell}_{i-1})m_{\rho[0]}(j) + q_{i-1}(\alpha^{\ell}_{i-1})m_{\rho[1]}(j)$
     $v_i(\alpha^{\ell}_{i-1}||j) \leftarrow v'_i(\alpha^{\ell}_{i-1}||j) + v_{i-1}(\alpha^{\ell}_{i-1})m_{\rho[0]}(j) + r_{i-1}(\alpha^{\ell}_{i-1})m_{\rho[1]}(j)$
   - Set
     $q_i(\alpha^{\ell}_{i-1}||j) \leftarrow q'_i(\alpha^{\ell}_{i-1}||j) + u_{i-1}(\alpha^{\ell}_{i-1}) \cdot p_{\rho[0]}(j) + q_{i-1}(\alpha^{\ell}_{i-1}) \cdot p_{\rho[1]}(j)$,
     $r_i(\alpha^{\ell}_{i-1}||j) \leftarrow r'_i(\alpha^{\ell}_{i-1}||j) + v_{i-1}(\alpha^{\ell}_{i-1}) \cdot p_{\rho[0]}(j) + r_{i-1}(\alpha^{\ell}_{i-1}) \cdot p_{\rho[1]}(j)$
6. Return $CW^i, s_i, t_i, u_i, v_i, q_i, r_i$

Figure 8: GenNext computes the seed and bit components of nodes at the next tree layer, where G is a PRG that each take an input of size λ bits and outputs a bit string of length $2(\lambda + 2)$.

Eval, k-sparse DPF

$Eval(b, g, k_b, x, B, k)$:

1. Parse $k_0 = s_0(0)||CW^1||CW^2||...||CW^{d+1}||h_1||\ldots||h_{d+1}$. Let $u_0(0) = b \cdot (g == 0)$ and $q_0(0) = b \cdot (g == 1)$.
2. for i from 1 to d:
   - Denote x_{i-1} the $(i-1)$-bit prefix of x. Also denote $\rho = h_i(x_{i-1})$.
   - Parse $CW^i = c_0||m_0(0)||p_0(0)||m_0(1)||$ $p_0(1)||...||c_{ck-1}||m_{ck-1}(0)||p_{ck-1}(0)||m_{ck-1}(1)||p_{ck-1}(1)$
   - Expand $\tau^i \leftarrow G(s_{i-1}(x_{i-1}))$
   - Set $CW^i_{\rho[0]} = c_{\rho[0]}||m_{\rho[0]}(0)||p_{\rho[0]}(0)||c_{\rho[0]}||m_{\rho[0]}(1)||p_{\rho[0]}(1)$
   - Set $CW^i_{\rho[1]} = c_{\rho[1]}||m_{\rho[1]}(0)||p_{\rho[1]}(0)||c_{\rho[1]}||m_{\rho[1]}(1)||p_{\rho[1]}(1)$
   - Compute $\tau^i = \tau^i \oplus u_{i-1}(x_{i-1}) \cdot CW^i_{\rho[0]} \oplus q_{i-1}(x_{i-1}) \cdot CW^i_{\rho[1]}$
   - Parse $\tau^i = s_i(x_{i-1}||0)||u_i(x_{i-1}||0)||q_i(x_{i-1}||0)||$ $s_i(x_{i-1}||1)||u_i(x_{i-1}||1)||q_i(x_{i-1}||1) \in \{0,1\}^{2(\lambda+k)}$
3. Denote $i = d+1$ and x_{i-1} the $(i-1)$-bit prefix of x. Also denote $\rho = h_i(x_{i-1})$.
4. Parse $CW^{d+1} = c_0(0)||...||c_0(B-1)||...||c_{ck-1}(0)||...||c_{ck-1}(B-1)$
5. Expand and parse $s'_i(x_{i-1}||0)||...||s'_i(x_{i-1}||B-1) \leftarrow G'(s_d(x_{i-1}))$
6. Convert $s'_i(x_{i-1}||j) := convert(s'_i(x_{i-1}||j))$ for $j \in [B]$
7. For $j \in [B]$, compute $s_i(x_{i-1}||j) = s'_i(x_{i-1}||j) + u_{i-1}(x_{i-1}) \cdot c_{\rho[0]}(j) + q_{i-1}(x_{i-1}) \cdot c_{\rho[1]}(j)$
8. Return $(-1)^b \cdot s_i(x)$

Figure 9: Eval evaluates one path x given a DPF key k_b corresponding to server b for k-block-sparse vectors with block size B, where G and G' are PRGs that each take an input of size λ bits and output a bit string of length $2(\lambda+2)$ and $B \log |\mathbb{G}|$, respectively. g defines which correction word will be applied in the first layer. Also, let $convert$ be a function that takes as input a bit string of length $\log |\mathbb{G}|$ and outputs a group element in $\mathbb{G}$.

subtraction of the check-bits in the two PRG outputs together with the check-bits of the appropriate correction word (if any) are zero. Building on the notation of Section H, taking $x \in \{0,1\}^d$ to be a node in the final layer of the DPF tree, and taking $s^{chk}(x)$ and $t^{chk}(x)$ to be the check-bits of the PRG output on the node x for the two servers (respectively) and taking c^{chk}_m to be the check-bits of the m-th correction word, the servers check that for each node x in the final layer:

$$\begin{aligned} 0 = s^{chk}_d(x) - t^{chk}_d(x) - \Big((u_d(x) + v_d(x)) \cdot c^{chk}_{f_{d+1}(x)[0]} \Big) \\ - \Big((q_d(x) + r_d(x)) \cdot c^{chk}_{f_{d+1}(x)[1]} \Big) . \end{aligned}$$

To verify that equality to zero holds for all x simultaneously, the servers can take a random linear combination of their individual summands and check only that the linear combination equals zero (this boils down to computing a linear function over their secret shared values, the random linear combination can be derandomized by taking the powers of a random field element). This only requires exchanging a constant number of field elements.

This part of the proof maintains zero knowledge. The new information revealed to the servers are the check-bits in the correction words. The concern could be that these expose something about the locations or the values of non-zero blocks. However, the check bits of the correction words are pseudorandom even given all seeds held by a single server, and given all the payload values of all correction words, and thus each server's view can be simulated and zero-knowledge is maintained.

The above construction is appealing, but it is not quite sound: intuitively, given that there are only k active correction bits (within the k correction bit pairs), the correction words are only applied to at most k of the blocks. If the check passes, this means that the check bits of all but at most k of the blocks had to have been 0 (except for a small error probability in the choice of the linear combination). However, it might be the case that the check-bits are 0, but the payload is not: i.e. we have two PRG seeds whose outputs are identical in their λ bit suffix, but not in the prefix. One way to resolve this issue would be by assuming that the PRG is injective (in its suffix), or collision intractable. We prefer not to make such assumptions, and instead use an additional round of interaction to ensure that soundness holds (the interaction can be eliminated using the Fiat-Shamir heuristic). The interactive construction is as follows:

1. The client sends all information for the DPF except the correction words (payload and check bits) for the last layer (B group elements and λ bits per node).
2. The servers choose a pairwise-independent function h mapping the range of the last layer's PRG to the same space and reveal it to the client.
3. The client computes the correction words for the last layer, where the PRG used for that layer is the composition $(h \circ G')$ (the pairwise independent hash function applied to the PRG's output).

Zero-knowledge is maintained because $(h \circ G')$ is still a PRG. Soundness now holds because the seeds for the final layer are determined before h is chosen. The probability that two non-identical seeds collide in their last λ bits, taken over the choice of h, is $2^{-\lambda}$. We take a union bound over all the seed-pairs and soundness follows.

I Additional Details on Experiments

We provide more details about our experimental results in Section I.1 and provide additional experiments and plots in Section I.2.

I.1 Experimental setup for private model training

Here we provide the full details for our private training experiments that were used to produce the plots in Figure 5.

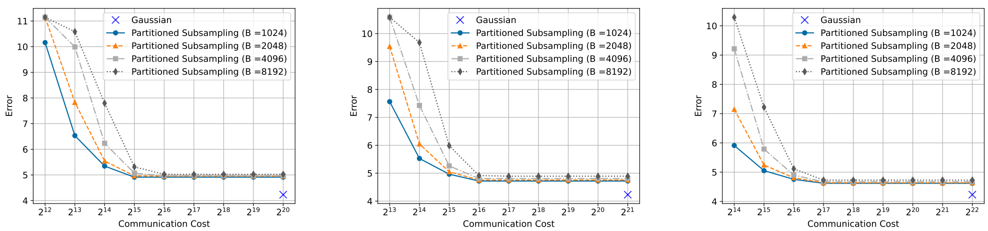
Noise comparison for DP-SGD (Figure 5a). For our experiment that compares the standard deviation of the noise of the Gaussian mechanism and our approach, we consider a private training setup where we have a model of size $D = 2^{20} \approx 10^6$ and number of data points $n = 6 \cdot 10^5$. We user a standard setting of the parameters of DP-SGD where we have clipping norm 1.0, $\varepsilon = 1.0$, $\delta = 10^{-6}$, and number of epochs is 10. We calculate the standard deviation of the noise required for the Gaussian mechanism using the dp-accounting library in python. For our approach, we calculate the standard deviation based on our description in the paper using the accounting techniques of [FS25]. For our method, we fix the communication cost $k \cdot B = 32768$ and then plot the ratio of the noise needed for our method over the noise of the Gaussian mechanism as a function of the block size B. We repeat this for different values of batch size in $\{512, 1028, 4096, 6 \cdot 10^5\}$ and report the results in Figure 5a.

MNIST experiment (Figure 5b). For MNIST, we follow the experimental setup of [AFN+23] and train a neural network with 69050 parameters (see full description in Table 1). We run DP-SGD with fixed learning rate 0.1, momentum 0.9, and batch size 600 for 10 epochs. To privatize the gradients at each batch, we clip each individual gradient to have ℓ_2 norm at most 1 and use the standard Gaussian mechanism or our partitioned subsampling approach to release private gradients. We calculate the standard deviation of the noise required for the Gaussian mechanism using the dp_accounting library in python. We set our privacy parameters to be $\varepsilon = 2.0$ and $\delta = 10^{-6}$. For our partitioned subsampling approach, we use a block size $B = 920$ and number of blocks $k = 20$, and clip the ℓ_2 norm of each block to be at most $L = 1.02\sqrt{B/D}$ where $D = 69050$ is the number of parameters in the model. We repeat this process 10 times, each time recoding the accuracy per epoch for each method, and plot the median accuracy with 90% confidence intervals in Figure 5b.

CIFAR10 experiment (Figure 5c). For CIFAR, we produce CLIP embedding (using the version ViT-B/32) for the CIFAR10 images and train a simple two-layer neural network with 66954 parameters: our network is a sequence of two fully connected layers: the first has dimensions 512×128 and the second 128×10. Then, we run DP-SGD with initial learning rate 4.0, momentum 0.9, weight decay $5 \cdot 10^{-4}$ and full batch for 10 epochs. We use a stepLR scheduler for the learning rate which reduces the learning rate by a factor of 0.9 every 5 epochs. Similarly to MNIST, we use the Gaussian mechanism and our partitioned subsampling approach to release private gradients, where we have clipping norm 1 for the gradients, $\varepsilon = 2.0$ and $\delta = 10^{-6}$. For our partitioned subsampling

Layer	Parameters
Convolution + $\tanh$	16 filters of 8×8, stride 2, padding 2
Average pooling	2×2, stride 1
Convolution + $\tanh$	32 filters of 4×4, stride 2, padding 0
Average pooling	2×2, stride 1
Fully connected + $\tanh$	32 units
Fully connected + $\tanh$	10 units

Table 1: Architecture for convolutional network model.



(a) (b) (c)

Figure 10: The trade-off between the standard deviation of the error (per coordinate) and per-client communication $C = kB$, when computing the sum of $n = 10^5$ vectors with dimension (a) $D = 2^{20}$, (b) $D = 2^{21}$, and (c) $D = 2^{22}$, with $(1.0, 10^{-6})$-DP. The blue 'x' shows the baseline approach of sending the whole vector.

approach, we use a block size $B = 920$ and number of blocks $k = 25$, and clip the ℓ_2 norm of each block to be at most $L = 1.02\sqrt{B/D}$ where $D = 66954$ is the number of parameters in the model. We repeat this process 10 times, each time recoding the accuracy per epoch for each method, and plot the median accuracy with 90% confidence intervals in Figure 5c.

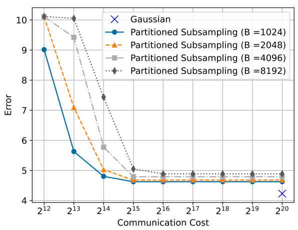
All of our experiments were run locally on a Macbook Pro equipped with Apple M1 Pro chip (with 10 cores), and 32GB RAM. The time for each epoch depends on the mechanism, dataset choice and batch size. For the Gaussian mechanism, each epoch takes from a few seconds up to 30 seconds, while for the partitioned subsampling approach each epochs takes about 1-2 minutes.

I.2 Additional experiments

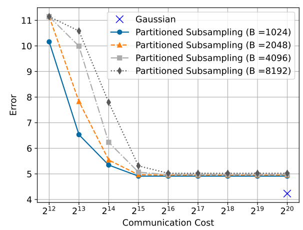
In this section, we present additional experimental results in different regimes than the ones presented in the main paper. We begin in Fig. 10 where we compare our approach to the Gaussian mechanism and plot the error for estimating the sum of unit vectors in different dimensions. We can see that even for small communication complexity, sometime a factor of 32 smaller than the dimension, our approach becomes competitive with the Gaussian mechanism. Fig. 11 presents a similar plot where we show that the same behavior holds for different number of samples n.

In Fig. 12 we compare the performance of the partitioned subsampling scheme and the truncated Poisson, where the plots show that each method is favorable in different regimes: the truncated Poisson obtains better error if more communication is allowed, getting closer to the error of the Gaussian mechanism. This is partly due to the analysis of partitioned subsampling building on RDP analysis, which even for the Gaussian mechanism yields standard deviation bounds slightly larger than the analytic Gaussian mechanism. The truncated Poisson analysis uses the tighter PRV accounting.

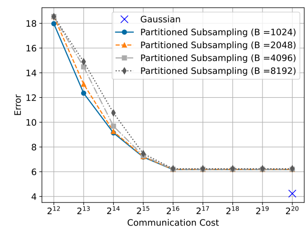
Furthermore, we evaluate our method for estimating the mean of real data. Specifically, we compare our method to the Gaussian mechanism for estimating the average gradient in a particular epoch during the training of a model over the CIFAR10 dataset with CLIP embeddings. We save 1024 gradients, each of dimension $D = 66954$, and employ our alternative method to estimate the average gradient under $(2.0, 10^{-6})$-DP. We present the results in Fig. 13. These results corroborate our findings in the main paper for synthetic data (see Fig. 4c), demonstrating that the same behavior is observed for realistic data.



(a)

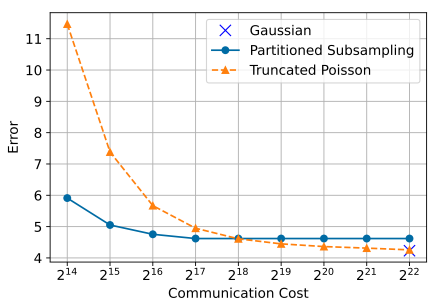


(b)

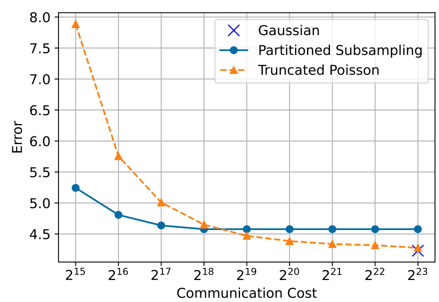


(c)

Figure 11: The trade-off between the standard deviation of the error (per coordinate) and per-client communication $C = kB$, when computing the sum of $D = 2^{20}$-dimensional vectors with sample size (a) $n = 10^4$, (b) $n = 10^5$, and (c) $n = 10^6$, with $(1.0, 10^{-6})$-DP. The blue 'x' shows the baseline approach of sending the whole vector.

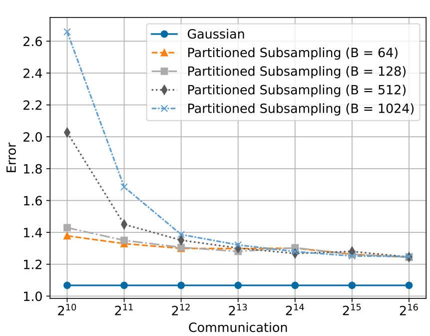


(a)

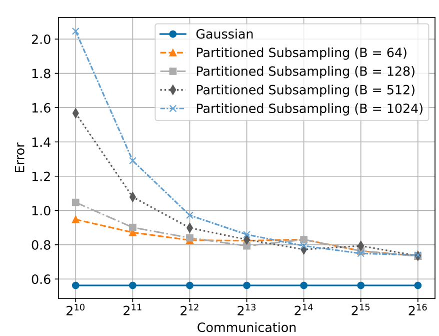


(b)

Figure 12: The trade-off between the standard deviation of the error (per coordinate) and per-client communication for the Partitioned Subsampling scheme and the Truncated Poisson. These plots are for aggregating $n = 10^5$ vectors with dimension (a) $D = 2^{22}$, (b) $D = 2^{23}$, block size $B = 2^{10}$ and $(1.0, 10^{-6})$-DP.



(a)



(b)

Figure 13: Error for estimating the average gradient during private model training of the CIFAR10 experiment where the model has $D = 66954$ parameters, comparing the Gaussian mechanism and partitioned subsampling for different communication costs. We use block size $B = 920$ for partitioned subsampling, and privacy parameter (a) $\varepsilon = 1$ and (b) $\varepsilon = 2$.

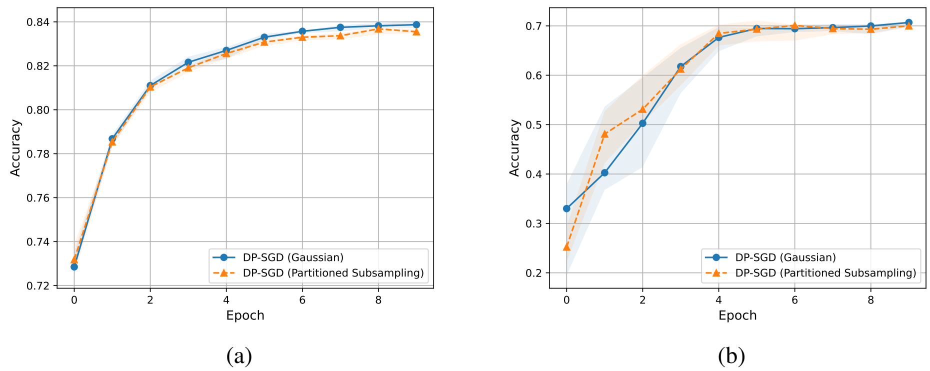


Figure 14: Comparison between PREAMBLE and the Gaussian mechanism for private model training over the CIFAR10 dataset using CLIP embeddings (version RN50 or Resnet50). We plot 90% confidence intervals for (a) batch size of 1024 and (b) full batch.

Finally, in Fig. 14, we present additional experiments for private model training on CIFAR10 using CLIP embeddings, employing the ResNet50 architecture. This experiment adheres to the same setup and parameters as Fig. 5c. Furthermore, we experiment with a batch size of 1024 and a learning rate of 0.5 (while maintaining all other parameters at the same values). We run each method 5 times and report the median accuracy as a function of epoch. Our plots demonstrate that our method performs similarly to the Gaussian mechanism for a small batch size and full batch, with a significant reduction in communication.

J Broader Impact

This paper presents work whose goal is to advance the field of Differentially Private Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.