Private Synthetic Data Generation in Bounded Memory

RAYNE HOLLAND, SEYIT CAMTEPE, CHANDRA THAPA, and MINHUI XUE, CSIRO's Data61, Australia

Protecting sensitive information on data streams is a pivotal challenge for modern systems. Current approaches to providing privacy in data streams can be broadly categorized into two strategies. The first strategy involves transforming the stream into a private sequence of values, enabling the subsequent use of non-private methods of analysis. While effective, this approach incurs high memory costs, often proportional to the size of the database. Alternatively, a compact data structure can be used to provide a private summary of the stream. However, these data structures are limited to predefined queries, restricting their flexibility.

To overcome these limitations, we propose a lightweight synthetic data generator, PrivHP, that provides differential privacy guarantees. PrivHP is based on a novel method for the private hierarchical decomposition of the input domain in bounded memory. As the decomposition approximates the cumulative distribution function of the input, it serves as a lightweight structure for synthetic data generation. PrivHP is the first method to provide a principled trade-off between accuracy and space for private hierarchical decompositions. It achieves this by balancing hierarchy depth, noise addition, and selective pruning of low-frequency subdomains while preserving high-frequency ones, all identified in a privacy-preserving manner. To ensure memory efficiency, we employ private sketches to estimate subdomain frequencies without accessing the entire dataset.

Central to our approach is the introduction of a pruning parameter k, which enables an almost smooth interpolation between space usage and utility, and a measure of skew $\texttt{tail}_k$, which is a vector of subdomain frequencies containing all but the largest k coordinates. PrivHP processes a dataset $\mathcal{X}$ using $M = O(k \log^2 |\mathcal{X}|)$ space and, on input domain $\Omega = [0,1]^d$, while maintaining ε-differential privacy, produces a synthetic data generator that is at distance

$$O\left(\frac{M^{(1-\frac{1}{d})}}{\varepsilon n} + \frac{||\texttt{tail}_k(\mathcal{X})||_1}{M^{1/d} n}\right)$$

from the empirical distribution in the expected Wasserstein metric. Compared to the state-of-the-art, PMM, which achieves accuracy $O((\varepsilon n)^{-1/d})$ with memory $O(\varepsilon n)$, our method introduces an additional approximation error term of $O(||\texttt{tail}_k(\mathcal{X})||_1/(M^{1/d} n))$, but operates in significantly reduced space. Additionally, we provide interpretable utility bounds that account for all error sources, including those introduced by the fixed hierarchy depth, privacy noise, hierarchy pruning, and frequency approximations.

CCS Concepts: • **Security and privacy → Data anonymization and sanitization**.

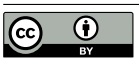
Additional Key Words and Phrases: Sketching, Differential Privacy, Synthetic Data

ACM Reference Format:
Rayne Holland, Seyit Camtepe, Chandra Thapa, and Minhui Xue. 2025. Private Synthetic Data Generation in Bounded Memory. *Proc. ACM Manag. Data* 3, 2 (PODS), Article 107 (May 2025), 25 pages. https://doi.org/10.1145/3725244

1 Introduction

A data stream solution addresses the problem of analyzing large volumes of data with limited resources. Numerous techniques have been developed to enable real-time analytics with minimal

Authors' Contact Information: Rayne Holland, rayne.holland@data61.csiro.au; Seyit Camtepe, seyit.camtepe@data61.csiro.au; Chandra Thapa, chandra.thapa@data61.csiro.au; Minhui Xue, jasox.xue@data61.csiro.au, CSIRO's Data61, Australia.



ACM 2836-6573/2025/5-ART107
https://doi.org/10.1145/3725244

memory usage and high throughput [7, 16, 21]. However, if the stream contains sensitive information, privacy concerns become paramount [12, 29]. In such cases, a data stream solution must optimize resource efficiency while balancing utility and strong privacy protection.

The concept of *differential privacy* has emerged as the prevailing standard for ensuring privacy in the analysis of data streams. It guarantees that an observer analyzing the outputs of a differentially private algorithm is fundamentally limited, in an information-theoretic sense, in their ability to infer the presence or absence of any individual data point in the stream. Methods for supporting differentially private queries on data streams can be broadly categorized into two strategies. The first strategy transforms the data stream into a differentially private sequence of values or statistics, which can subsequently be processed and queried by non-private data structures [6, 17, 23, 29]. However, these approaches can incur high memory costs, limiting their applicability in many streaming contexts. The second approach involves constructing specialized data structures, with small memory allocations, that provide differentially private answers to specific, predefined queries [3, 18, 33]. Although this approach is memory efficient, it restricts the range of queries to those selected in advance, reducing flexibility.

To address these limitations, we propose a lightweight synthetic data generator, `PrivHP` (Private Hot Partition), that provides differential privacy and operates with high throughput and in bounded memory. Our generator produces private synthetic data that approximates the distribution of the original data stream. This synthetic data can be used for any downstream task without additional privacy costs. Therefore, `PrivHP` both protects sensitive information in bounded memory *and* supports a broad range of queries in resource-constrained environments.

1.1 Problem and Solution

Mathematically, the problem of generating synthetic data can be defined as follows. Let (Ω, ρ) be a metric space and consider a stream $\mathcal{X} = (X_1, \ldots, X_n) \in \Omega^n$. Our goal is to construct a space and time-efficient randomized algorithm that outputs synthetic data $\mathcal{Y} = (Y_1, \ldots, Y_m) \in \Omega^m$, such that the two empirical measures

$$\mu_{\mathcal{X}} = \frac{1}{n}\sum_{i=1}^{n}\delta_{X_i} \quad \text{and} \quad \mu_{\mathcal{Y}} = \frac{1}{m}\sum_{i=1}^{m}\delta_{Y_i}$$

are close together. Moreover, the output $\mathcal{Y}$ should satisfy differential privacy.

Our approach adapts recent advancements in private synthetic data generation (in a non-streaming setting) that utilize a *hierarchical decomposition* to partition the sample space [14, 15]. At a high level, these methods work in the following way:

(1) Hierarchically split the sample space into smaller subdomains;
(2) For each subdomain, count how often items from the dataset appear within it;
(3) To ensure privacy, add a carefully chosen amount of random noise to each frequency count;
(4) Use these noisy counts to construct a probability distribution so that items from each subdomain can be sampled with a probability proportional to their noisy frequency.

The accuracy of this approach depends on the granularity of the partition. Creating more subdomains of smaller diameter can improve the approximation of the real distribution, but requires more memory. Therefore, the challenge of constructing a high-fidelity partition on a stream is this balance between granularity and memory.

To meet this challenge, `PrivHP` adopts a new method for hierarchical decomposition that *prunes* less significant subdomains while prioritizing subdomains with a high frequency of items. In addition, we use sketching techniques to approximate frequencies at deeper levels in the hierarchy,

allowing for effective pruning without requiring access to the entire dataset. Similar to He *et al.* [15], to maintain privacy, we perturb the frequency counts of nodes in the (pruned) hierarchy.

1.2 Main Result

We measure the utility of a synthetic data generator $\mathcal{T}$ by $\mathbb{E}[W_1(\mu_X, \mathcal{T})]$, where $W_1(\mu_X, \mathcal{T})$ is the 1-Wasserstein metric, and $\mathbb{E}$ is taken over the randomness of the algorithm generating $\mathcal{T}$. Our results explore trade-offs between utility and performance. We are interested in quantifying the cost, in utility, of supporting synthetic data generation under resource constraints. To quantify the cost of pruning we introduce the vector $\texttt{tail}_k^r$, which is the vector of subdomain frequencies, at level r in the hierarchy, with the highest k coordinates set to 0. The norm $||\texttt{tail}_k^r||_1$ is small for skewed inputs and can even be zero on sparse inputs. Our utility bound, expressed as a function of the memory allocation, is formalized in the following result on the hypercube $\Omega = [0,1]^d$.

Theorem 1.1. *When $\Omega = [0,1]^d$, for pruning parameter k,* PrivHP *can process a stream X of size n in $M = O(k \log^2(n))$ memory and $O(\log(\varepsilon n))$ update time.* PrivHP *can subsequently output a ε-differentially private synthetic data generator $\mathcal{T}_{\texttt{PrivHP}}$, in $O(M \log n)$ time, such that*

$$\mathbb{E}[W_1(\mu_X, \mathcal{T}_{\texttt{PrivHP}})] = \begin{cases} O\left(\frac{\log^2(M)}{\varepsilon n} + \frac{||\texttt{tail}_k^{\varepsilon n}(X)||_1}{Mn}\right) & if\, d = 1 \\ O\left(\frac{M^{(1-\frac{1}{d})}}{\varepsilon n} + \frac{||\texttt{tail}_k^{\varepsilon n}(X)||_1}{M^{1/d} n}\right) & if\, d \geq 2 \end{cases},$$

Our work makes several key contributions. First, we present a new method for private hierarchical decomposition that provides the first known trade-off between accuracy and space. Our approach achieves this by carefully balancing hierarchical depth, noise addition, and pruning, allowing for efficient representations without sacrificing utility. The key innovation is the introduction of the pruning parameter k, which controls the accuracy-space trade-off and provides a level of flexibility unavailable in prior work. In addition, Theorem 1.1 reveals that, as the noise term[1] grows with the memory allocation, for highly skewed or sparse inputs, pruning may be beneficial in the non-streaming setting.

A central contribution of our work is the accuracy analysis. The key challenge lies in understanding how error (from both privacy noise and frequency approximation) propagates down the hierarchy and affects pruning decisions. We formalize how this error is propagated and provide interpretable utility bounds (Theorem 5.2) that account for the different sources of error.

These bounds apply to *any* metric space as an input domain. Thus, our techniques are applicable to a broad range of domains, such as geographic coordinates or the IPv4 address space.

Leveraging the constructed hierarchy, we introduce the first private synthetic data generator with provable space-utility trade-offs for any metric space. This enables the generation of high-fidelity, privacy-preserving synthetic datasets, in bounded memory, that are suitable for various downstream data analysis tasks and applications. A comprehensive comparison with existing methods for private synthetic data generation is provided in Table 1.

1.3 Organization of the Paper

As background, Section 2 covers related work and Section 3 establishes the relevant preliminaries. Section 4 introduces our method for compact hierarchical decompositions on the stream. Section 5 covers synthetic data generation using hierarchical partitions and supplies the main results. Section 6 introduces our method for measuring utility. Lastly, Section 7 contains the proof of our utility bound for general input domains.

[1]The noise term is $O(\log^2 M/(\varepsilon n))$ for the case $d = 1$

Method	Accuracy		Memory
	$\Omega = [0,1]$	$\Omega = [0,1]^d, d \geq 2$	
Smooth [30]	$O\left(\varepsilon^{-1} n^{-K/(2d+K)}\right)$		$O(dn)$
SRRW [5]	$O\left(\left(\log^{\frac{3}{2}}(\varepsilon n)(\varepsilon n)^{-1}\right)^{1/d}\right)$		$O(dn)$
PMM [15]	$O\left(\log^2(\varepsilon n)(\varepsilon n)^{-1}\right)$	$O\left((\varepsilon n)^{-1/d}\right)$	$O(\varepsilon n)$
PrivHP	$O\left(\frac{\log^2(M)}{\varepsilon n} + \frac{\|\|\mathtt{tail}_k^{\varepsilon n}\|\|_1}{Mn}\right)$	$O\left(\frac{M^{(1-\frac{1}{d})}}{\varepsilon n} + \frac{\|\|\mathtt{tail}_k^{\varepsilon n}\|\|_1}{M^{1/d} n}\right)$	$M = O(k \log^2 n)$

Table 1. Performance of PrivHP vs prior work. Results are presented for input domains $\Omega = [0,1]$ and $\Omega = [0,1]^d$ ($d \geq 2$). Accuracy is measured by the expected 1-Wasserstein distance. The utility guarantee for Smooth is restricted to smooth queries with bounded partial derivatives of order K.

2 Related Work

2.1 Private Hierarchical Decomposition

Many methods for hierarchical decomposition have been adapted to the context of differential privacy [9, 17, 24, 31, 32]. Static solutions, such as PrivTree [32], require full access to the dataset and are not suitable for streaming. The dynamic decomposition introduced by PrivStream [17] adapts PrivTree to the stream. However, PrivStream is supported by a fixed and potentially large hierarchical partition of the input domain that must be stored locally with exact counts [17]. Therefore, it incurs large memory costs and is not suited to resource-constrained settings. We overcome this limitation by adopting pruning and sketching techniques to summarize deeper levels of the hierarchy. In addition, PrivStream does not provide any utility guarantees.

Biswas *et al.* introduced a streaming solution for hierarchical heavy hitters [3] that can support a private hierarchical decomposition on the stream. One key difference in our approach is the choice of private sketch. The hashing-based private sketch [22] employed by PrivHP has a better error guarantee than the counter-based sketch [18] used by Biswas *et al.* [3]. Further, as the error of the hash-based sketch can be expressed in terms of the tail of the dataset it composes nicely with hierarchy pruning.

2.2 Privacy on Data Streams

A common strategy for supporting privacy on streams is to transform the stream into a differentially private sequence of values [6, 23, 29]. While this enables query flexibility, current methods provide no bounds on the memory allocation, which can be proportional to the size of the stream. This limits their application in resource-constrained environments. In the traditional data steam model, where memory is sublinear in the size of the database, the current approach for protecting streams is to construct specialized data structures, with small memory allocations, that provide differentially private answers to specific, predefined queries [3, 18, 33]. However, these methods lack query flexibility.

Alabi *et al.* provide a method for private quantile estimation in bounded memory [1]. A quantile estimator can be used to generate synthetic data approximating the distribution of the input dataset. This is achieved by sampling a value uniformly in [0,1] and returning the quantile. However, their method only works for finite and ordered input domains and, thus, does not extend to general metric spaces.

2.3 Non-Streaming Private Synthetic Data Generation

The problem of generating private synthetic data from *static* datasets, especially in relation to differential privacy, has been explored in depth. The challenge of this problem was demonstrated by Ullman and Vadhan, who showed that, given assumptions on one-way functions, generating private synthetic data for all two-dimensional marginals is NP-hard on the Boolean cube [26]. A subsequent portion of research has since focused on guaranteeing privacy for specific query sets [2, 4, 10, 19, 25, 27].

The utility for private synthetic data is measured by the expected 1-Wasserstein distance. Wang *et al.* [30] addressed private synthetic data generation on the hypercube $[0, 1]^d$. They introduced a method (`Smooth`) for generating synthetic data that comes with a utility guarantee for smooth queries with bounded partial derivatives of order K, achieving accuracy $O(\varepsilon^{-1} n^{-K/(2d+K)})$. More recently, Boedihardjo *et al.* [5] proved an accuracy lower bound of $O(n^{-1/d})$. They also introduced an approach (`SRRW`) based on super-regular random walks with near-optimal utility of $O(\log^{3/2}(\varepsilon n)(\varepsilon n)^{-1/d})$. Subsequently, He *et al.* [15] proposed an approach `PMM`, based on hierarchical decomposition, that achieves optimal accuracy (up to constant factors) for $d \geq 2$. These methods provide a combination of privacy and provable utility. However, they do not consider resource constraints. In contrast, our approach provides meaningful trade-offs between utility and resources. A summary of accuracy vs. memory for prior work is presented in Table 1.

3 Preliminaries

3.1 Privacy

Differential privacy ensures that the inclusion or exclusion of any individual in a dataset has a minimal and bounded impact on the output of a mechanism. Two streams $\mathcal{X}$ and $\mathcal{X}'$ are *neighboring*, denoted $\mathcal{X} \sim \mathcal{X}'$, if they differ in one element. Formally, $\mathcal{X} \sim \mathcal{X}'$ if there exists a unique i such that $x_i \neq x'_i$. The following definition of differential privacy is adapted from Dwork and Roth [11].

Definition 3.1 (Differential Privacy – 1-Pass). A randomized mechanism $\mathcal{M}$ satisfies ε-differential privacy in a 1-*pass* setting if and only if, for all pairs of neighboring streams $\mathcal{X} \sim \mathcal{X}' \in \Omega^*$ and all measurable sets of outputs $Z \subseteq \texttt{support}(\mathcal{M})$, it holds that

$$\Pr[\mathcal{M}(\mathcal{X}) \in Z] \leq e^{\varepsilon} \Pr[\mathcal{M}(\mathcal{X}') \in Z].$$

This states that the output after the stream is processed is differentially private. This is in contrast to continual observation, where the output is published after each stream update. Our focus is on the 1-pass model, but our method can be adapted to continual observation by replacing the counters and sketches with their continual observation counterparts.

The Laplace mechanism is a fundamental technique for ensuring differential privacy by adding noise calibrated to a function's sensitivity. Let $\triangle_p(f) = \max_{\mathcal{X} \sim \mathcal{X}'} \|f(\mathcal{X}) - f(\mathcal{X}')\|_p$ denote the p-sensitivity of the function f.

Lemma 3.2 (Laplace Mechanism). *Let f be a function with L_1-sensitivity $\triangle_1(f)$. The mechanism*

$$\mathcal{M}(\mathcal{X}) = f(\mathcal{X}) + \texttt{Laplace}\left(\frac{\triangle_1(f)}{\varepsilon}\right),$$

satisfies ε-differential privacy, where `Laplace` *is a Laplace distribution with mean 0 and scale parameter $\frac{\triangle_1(f)}{\varepsilon}$.*

3.2 Utility

The utility of the output is measured by the expectation of the 1-Wasserstein distance between two measures:

$$\mathcal{W}_1(\mu_X, \mu_Y) = \sup_{\text{Lip}(f)\leq 1} \left(\int f d\mu_X - \int f d\mu_Y \right), \tag{1}$$

where the supremum is taken over all 1-Lipschitz functions on Ω. Since many machine learning algorithms are Lipschitz [20, 28], Equation 1 provides a uniform accuracy guarantee for a wide range of machine learning tasks performed on synthetic datasets whose empirical measure is close to μ_X in the 1-Wasserstein distance.

3.3 Sketching

Our generator relies on a partition of the sample space. Prior works, such as `PrivTree` and `PMM`, construct partitions using *exact* frequency counts, which require access to the full dataset. Under the constraint of a sublinear memory allocation, we instead use approximate frequency counts that are private. We adopt private hash-based sketches [22, 33] to meet these needs.

A sketch performs a random linear transformation A to embed a vector $v \in \mathbb{R}^n$ into a smaller domain $Av \in \mathbb{R}^{j\times w}$. When the embedding dimension $j \times w$ is small, memory is reduced at the cost of increased error. A *private* sketch adds noise to the transformation to make the distribution of the sketch indistinguishable on neighboring inputs. Common examples are the Private Count Sketch and Private Count-Min Sketch [22, 33]. They share a similar structure but differ in their update and query procedures. We first overview non-private variants of sketches and then demonstrate how to apply differential privacy.

The Count-Min Sketch [8] is a $j \times w$ matrix of counters, $C(v) \in \mathbb{R}^{j\times w}$, determined by random hash functions $h_1, \ldots, h_j : [n] \to [w]$, where each h_i maps entries $x \in v$ into a bucket $h_i(x)$ within row i. For $(i, k) \in [j] \times [w]$, each bucket is defined as:

$$C(X)_{i,k} = \sum_{x\in v} v_x \cdot \mathbb{1}(h_i(x) = k),$$

where $\mathbb{1}(\xi)$ indicates event ξ and v_x is the count at entry x in v. Thus, each entry x is added to buckets $(i, h_i(x))$ for all $i \in [j]$, creating j hash tables of size w. The update procedure is visualized in Figure 1. The estimator $\hat{v}_x = \min\{C[i][h_i(x)] \mid i \in [j]\}$ combines row estimates by taking the minimum value, filtering out collisions with high-frequency items. Our results use the following bound on the expected error in a Count-min Sketch, where the influence of high frequency items decays exponentially with the number of rows.

Lemma 3.3. *For input vector v, the estimation error of a Count-min Sketch, with width $2w$ and depth j, satisfies, $\forall x \in v$,*

$$\mathbb{E}[\hat{v}_x - v_x] \leq \frac{||\texttt{tail}_w(v)||_1 + 2^{-j+1}||v||_1}{w}$$

where $\text{tail}_w(v)$ *is the vector v with the w largest coordinates removed.*

The proof can be found in Appendix A. Similar to prior work [22], our result relies on the use of fully random hash functions. Significantly, our privacy guarantee does not depend on this assumption.

3.4 Private Release of Sketches

For private release, sketches must have similar distributions on neighboring vectors. In *oblivious* approaches, we sample a random vector $v \in \mathbb{R}^{j\times w}$ independent of the data and release $C(X) + v$.

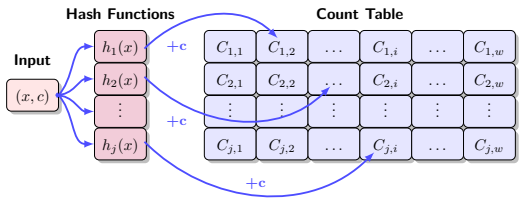


Fig. 1. The update procedure for a Count-Min Sketch. It follows that $h_j(x) = i$.

The sampling distribution depends on the sensitivity of the sketch. Since sketches are linear, for neighboring inputs $X \sim X'$, we have $C(X) - C(X') = C(X - X')$. As neighboring inputs have sensitivity $\triangle_1(X - X') = 1$, a sketch C has sensitivity proportional to its number of rows. Therefore, $\triangle_1(C) = j$ and $C(X) + v$ achieves ε-differential privacy for $v \sim \texttt{Laplace}^{j \times w}(j\varepsilon^{-1})$ by Lemma 3.2.

4 Private Hierarchical Decomposition in Bounded Memory

A hierarchical decomposition recursively splits a sample space into smaller subdomains. Each point of splitting refers to a *level* in the hierarchy. Formally, for a binary partition of the sample space Ω, the first level of the hierarchy contains disjoint subsets $\Omega_0, \Omega_1 \subset \Omega$, such that $\Omega_0 \cup \Omega_1 = \Omega$. Accordingly, a hierarchical decomposition $\mathcal{T}$ of depth L is a family of subsets Ω_θ indexed by $\theta \in \mathcal{T} \subseteq \{0,1\}^{\leq L}$, where

$$\{0,1\}^{\leq L} := \{0,1\}^0 \cup \{0,1\}^1 \cup \cdots \cup \{0,1\}^L.$$

By convention the cube $\{0,1\}^0 = \varnothing$. For Ω_θ with $\theta \in \{0,1\}^l$, we call l the level of Ω_θ. The leaves of the decomposition $\mathcal{T}$ form a partition of the sample space. To generate synthetic data, any decomposition $\mathcal{T}$ can be used to form a sampling distribution. A synthetic point can be constructed by (1) selecting a leaf subdomain $\Omega_\theta \subseteq \Omega$ with probability proportional to its cardinality and (2) conditioned on this selection, choosing a point $y \in \Omega_\theta$ uniformly at random.

Our lightweight generator is based on a *pruned* hierarchical decomposition. The quality of the generator depends on the granularity of the subsets in the partition. That is, allowing more subsets of smaller area leads to a sampling distribution closer to the empirical distribution of the input. Thus, due to the memory cost of storing a more fine-grained partition, we observe a trade-off between utility and space. To balance this trade-off, we aim to construct a hierarchical decomposition that provides finer granularity for "hot" parts of the sample space, where hot indicates a concentration of points. The high-level strategy is to branch the decomposition at hot nodes in the hierarchy. In order to bound the memory allocation, we introduce a *pruning* parameter k, which denotes the number of branches at each level of the hierarchy. Increasing k allows for more branches and, thus, finer granularity at the cost of more memory.

In addition, as the space occupied by the generator is sublinear in the size of the database, we cannot rely on exact frequency counts to compute cardinalities for every *possible* node in the decomposition. Therefore, we employ private sketches at deeper, and more populated, levels in the hierarchy to support approximate cardinality counting. For example, at level l, a single private sketch can be used to count the number of points in each subdomain Ω_θ, for $\theta \in \{0,1\}^l$. Once the private sketches process the data, they are then used to inform and grow the decomposition. That is, nodes in the hierarchy are considered hot if their noisy approximate counts are large. As the sketches are private, which means they are indistinguishable on neighboring inputs, the resulting decomposition is also private by the principle of post processing.

Algorithm 1: 1-pass PrivHP algorithm.

Input: Database $\mathcal{X}$ of items from Ω, pruning parameter k and privacy parameters $\{\sigma_l\}$.

1 **define** PrivHP$(\mathcal{X}, (k, L_\star, L), (w, j), \{\mathcal{D}_l\})$
  // Initialize Data Structures
2 Initialize complete binary partition tree $\mathcal{T}$ with depth $L_\star$
3 **for** $l \in \{0, \ldots, L_\star\}$ **do**
4 **for** $v \in \{v \in \mathcal{T} \mid \texttt{level}(v) = l\}$ **do**
5 $g \leftarrow$ random value drawn from $\mathcal{D}_l$
6 $v.\texttt{count} \leftarrow g$
7 **for** $l \in \{L_\star + 1, \ldots, L\}$ **do**
8 $\texttt{sketch}_l \leftarrow$ initialize private sketch with dimension (w, j) and noise $\mathcal{D}_l$
  // Parse Dataset
9 **for** $x \in \mathcal{X}$ **do**
10 **for** $l \in \{0, \ldots, L\}$ **do**
11 $\theta \leftarrow$ the unique $\theta' \in \{0,1\}^l$ such that $x \in \Omega_{\theta'}$
12 **if** $l \leq L_\star$ **then**
13 Increment the counter in $v_\theta \in \mathcal{T}$
14 **else**
15 $\texttt{sketch}_l.\texttt{update}(\theta, 1)$
16 $\mathcal{T}_{\texttt{PrivHP}} \leftarrow \texttt{GrowPartition}(\mathcal{T}, \{\texttt{sketch}_l\}, k)$ // Algorithm 2
17 **return** $\mathcal{T}_{\texttt{PrivHP}}$

To describe this process in more detail, we break it down into three components: initialization; parsing the data; and growing the partition.

4.1 Initialization

The boundaries of the subdomains Ω_θ can chosen arbitrarily. However, these boundaries must be fixed a priori. The pseudocode for initializing the component data structures is available in Algorithm 1 (Lines 2-8). The decomposition of the domain Ω is encoded in a binary tree $\mathcal{T}$, where each node in the tree $v_\theta \in \mathcal{T}$ represents the subset Ω_θ. Let $\texttt{level}(v_\theta)$ represent the level to which v_θ belongs.

The memory-utility trade-off for PrivHP is parameterized by $L_\star$, the level at which pruning begins and k, the number of branches at each level $l > L_\star$. The initial decomposition contains all subsets Ω_θ, for $\theta \in \{0,1\}^{\leq L_\star}$. Therefore, the algorithm begins by initializing $\mathcal{T}$ as a *complete* binary tree of depth $L_\star$ (Line 2).

For a decomposition of depth L, the sampling distribution of the generator is based on the cardinalities of each subset Ω_θ for $\theta \in \{0,1\}^{\leq L}$. We store noisy *exact* counts for subdomains Ω_θ, where $\theta \in \{0,1\}^{\leq L_\star}$, included in $\mathcal{T}$ and noisy *approximate* counts for subdomains Ω_θ, where $\theta \in \{0,1\}^{\leq L} \setminus \{0,1\}^{\leq L_\star}$. The exact counters are stored in their corresponding nodes in $\mathcal{T}$. To ensure privacy, each counter at level $l \leq L_\star$ is initialized with some random noise from a distribution $\mathcal{D}_l$ provided as input (Line 6). For each level $l > L_\star$, the approximate counts for subsets Ω_θ, with $\theta \in \{0,1\}^l$, are stored in a private sketch ($\texttt{sketch}_l$) of dimension $w \times j$ and initialized with random noise from the distribution $\mathcal{D}_l$ (Line 8). These sketches will be used to grow the decomposition (Line 16) according to hot subsets of the sample space once the stream has been processed.

Algorithm 2: Growing the PrivHP based on approximate counts at each level in the hierarchy.

Input: Partition Tree $\mathcal{T}$ and the collection of level-wise sketches $\{\texttt{sketch}_l\}$

1 **define** GrowPartition($\mathcal{T}, \{\texttt{sketch}_l\}, k$)
2 Apply consistency to each non-leaf node $v_\theta \in \mathcal{T}$ in depth-first order using Algorithm 3
3 $V \leftarrow \{\theta \mid v_\theta \in \mathcal{T}, \texttt{level}(v_\theta) = L_\star\}$
4 **for** $l \in \{L_\star + 1, \ldots, L-1\}$ **do**
5 **for** $\theta \in V$ **do**
6 **for** $\theta^* \in \{\theta 0, \theta 1\}$ **do**
7 $\hat{f}_{\theta^*} \leftarrow \texttt{sketch}_l.\texttt{query}(\theta^*)$
8 add v_{θ^*} to $\mathcal{T}$ with $v_{\theta^*}.\texttt{count} = \hat{f}_{\theta^*}$
9 Apply consistency to v_θ // Algorithm 3
10 $V \leftarrow$ the IDs of the Top-k values in $\{v.\texttt{count} \mid v \in \mathcal{T}, \texttt{level}(v) = l+1\}$
11 **return** $\mathcal{T}$

4.2 Parsing the Data

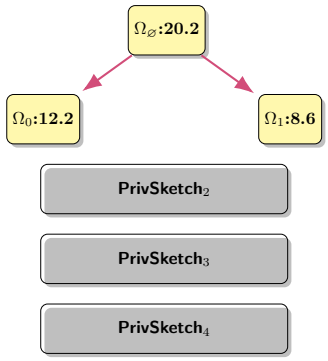
After initialization, we read the database in a single pass, one item at a time, while updating the internal data structures $\mathcal{T}$ and $\{\texttt{sketch}_l\}_{l>L_\star}$. For each update $x \in \mathcal{X}$, the procedure iterates through the levels in the hierarchy. At each level l, the unique subdomain $\theta \in \{0,1\}^l$, such that $x \in \Omega_\theta$, is identified. If $l \leq L_\star$, then the counter in node $v_\theta \in \mathcal{T}$ is updated. Otherwise, $\texttt{sketch}_l$ is updated with the subset index θ. At the end of the stream, $\mathcal{T}$ contains the noisy exact counts for subsets Ω_θ with $\theta \in \{0,1\}^{\leq L_\star}$ and the summaries $\{\texttt{sketch}_l\}$ contain the noisy approximate counts for subsets at levels $l > L_\star$. At this point, these approximate counts are used to grow $\mathcal{T}$ beyond level $L_\star$.

4.3 Growing the Partition

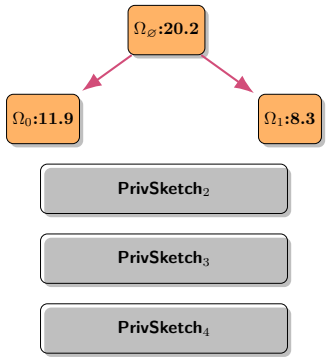
Pseudocode for this step is available in Algorithm 2, and an illustration of the process is provided in Figure 2. Before growing the partition, a consistency step is preformed. Consistency enforces two constraints. First, it requires that the count of a parent node equals the sum of the counts of its child nodes. Second, it requires that all counts are non-negative. After processing the data, $\mathcal{T}$ is not consistent due to the noise added for privacy. The outcome of this consistency step is presented in Figure 2b. An equivalent consistency step is common in private histograms [13], where it is observed it can increase utility at the same privacy budget.

After the consistency step has been executed, the partition is expanded one level at a time. The procedure begins by selecting the current leaf nodes V of $\mathcal{T}$ at level $L_\star$ (Line 3). These are considered "hot" nodes. Then, for each hot node $v_\theta \in V$, it adds the two child nodes ($v_{\theta 0}$ and $v_{\theta 1}$) to $\mathcal{T}$ as the decomposition of the node into two disjoint subsets (Figure 2c). In addition, the nodes $v_{\theta 0}$ and $v_{\theta 1}$ are initialized with the noisy frequency estimates retrieved from $\texttt{sketch}_{L_\star+1}$. These estimates are then adjusted according to the consistency step (Figure 2d).

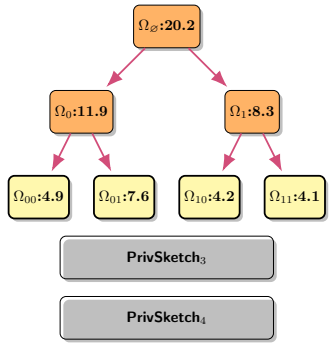
After all the hot nodes have been expanded, the next iteration of hot nodes needs to be selected. This is achieved by selecting the nodes with the Top-k frequency estimates (Line 10). With a new set of hot nodes, this process repeats itself and stops at depth $L-1$. In summary, at each level in the iteration, the current hot nodes are branched into smaller subdomains at the next level in the hierarchy. Then, the new subdomains with high frequency become hot at the subsequent iteration.



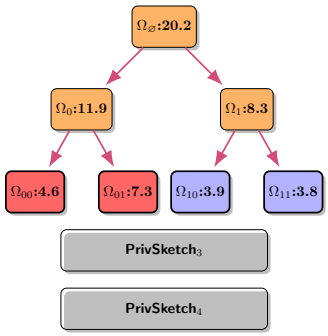
(a) After processing the stream



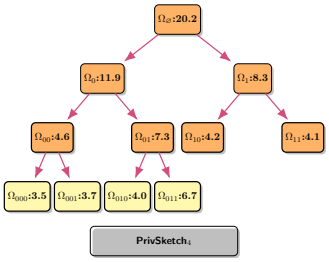
(b) After consistency is applied to the tree



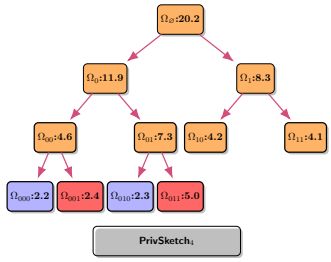
(c) Adding nodes from sketch$_2$



(d) After consistency is applied to level 2



(e) Pruning based on top-2 selection



(f) After consistency is applied to level 3

Fig. 2. Illustration of Algorithm 2 with $k = 2, L_\star = 1$ and $L = 4$. Figure 2a represents its input.

4.4 Consistency

Consistency ensures that (1) all counts are non-negative and (2) that the counts of two subregions add to the count of their parent region. The consistency step in Algorithm 2 is general and, following He *et al.* [15], adheres to the following rule. In the case of a deficit, when the sum of the two subregional counts is smaller than the count of the parent region, both subregional counts should be increased. Conversely, in the case of a surplus, both subregional counts should be decreased. Apart from this requirement, we are free to distribute the deficit or surplus between the subregional counts.

Our main utility bound (Theorem 5.2) is based on a concrete instance of consistency. The method we adopt is presented in Algorithm 3. The main idea is to *evenly* redistribute the error generated from sibling subregions. To formalize this idea, let v_θ denote the parent node. First, we calculate the difference between the subregional counts and their parent region (Line 5): $\Lambda = v_{\theta 0}.\texttt{count} + v_{\theta 1}.\texttt{count} - v_\theta.\texttt{count}$. This difference is then evenly redistributed across the subregions (Line 12):

$$v_{\theta 0}.\texttt{count} \leftarrow v_{\theta 0}.\texttt{count} - \Lambda/2 \quad \text{and} \quad v_{\theta 1}.\texttt{count} \leftarrow v_{\theta 1}.\texttt{count} - \Lambda/2. \tag{2}$$

We also add two correction steps for whenever this approach might violate consistency. The first correction makes sure that both subregional counts are non-negative prior to applying consistency

Algorithm 3: Enforcing consistency between nodes in $\mathcal{T}$.

Input: Node $v_\theta \in \mathcal{T}$

1 **define** `EnforceConsistency`(v_θ)
2 **for** $\theta^* \in \{\theta 0, \theta 1\}$ **do**
3 **if** $v_{\theta^*}.\texttt{count} < 0$ **then**
 `// Error Correction Type 1`
4 $v_{\theta^*}.\texttt{count} \leftarrow 0$
5 $\Lambda \leftarrow v_{\theta 0}.\texttt{count} + v_{\theta 1}.\texttt{count} - v_\theta.\texttt{count}$
6 **if** $\min\{v_{\theta 0}.\texttt{count} - \Lambda/2, v_{\theta 1}.\texttt{count} - \Lambda/2\} < 0$ **then**
 `// Error Correction Type 2`
7 $(\theta_{\texttt{min}}, \theta_{\texttt{max}}) \leftarrow$ order (θ_0, θ_1) by their counters
8 $v_{\theta_{\texttt{min}}} \leftarrow 0$
9 $v_{\theta_{\texttt{max}}}.\texttt{count} \leftarrow v_\theta.\texttt{count}$
10 **else**
11 **for** $\theta^* \in \{\theta 0, \theta 1\}$ **do**
12 $v_{\theta^*}.\texttt{count} \leftarrow v_{\theta^*}.\texttt{count} - \Lambda/2$
13 **return**

(Line 3). The second involves applying a different redistribution method in the event that (2) violates consistency (Line 6). In this instance, the count of the violating node is set to 0 and its sibling node inherits the full count from its parent. Both correction steps *reduce* the amount of error in the subregion counts.

5 Private Synthetic Data

An item can be sampled from the decomposition tree $\mathcal{T}$ by selecting a number u uniformly in the range $[0, v_\varnothing.\texttt{count}]$, where $v_\varnothing$ is the root node of $\mathcal{T}$. Then, a root-to-leaf traversal of the tree is performed. At each node v_θ on the path, we retrieve the count from the left child $c \leftarrow v_{\theta 0}.\texttt{count}$. We branch left if $c \geq u$; otherwise, we branch right. When branching right, u is updated with $u \leftarrow u - v_\theta.\texttt{count}$. The final leaf node represents a subset of the sample space and we can return any item uniformly at random from this subset.

Note that this sampling algorithm can take any binary decomposition of Ω as input. This makes any tree $\mathcal{T}$ synonymous with a sampling distribution. Therefore, throughout the rest of the paper we often refer to $\mathcal{T}$ as a probability distribution. For the remainder of this section, we establish privacy and provide bounds on the utility of the generator $\mathcal{T}_{\texttt{PrivHP}}$ output by Algorithm 1, where utility is measured in the expected 1-Wasserstein metric.

5.1 Privacy

Algorithm 2 is completely deterministic. Therefore, if the inputs to Algorithm 2 are differentially private, then the resulting partition is differentially private by the post-processing property. The random perturbations introduced at initialization depend on the collection of noise distributions $\{\mathcal{D}_l\}$. They should provide sufficient noise such that the output distributions of $\mathcal{T}$ and $\{\texttt{sketch}_l\}$ on neighboring datasets are indistinguishable. The are many choices for $\{\mathcal{D}_l\}$ that impact both privacy and utility. Here is one example.

Theorem 5.1. *If the noise distributions have the following form:*

$$\mathcal{D}_l = \begin{cases} \texttt{Laplace}(\sigma_l^{-1}) & \text{for } l \le L_\star \\ \texttt{Laplace}^{w \times j}(j\sigma_l^{-1}) & \text{Otherwise} \end{cases} \tag{3}$$

Then, the decomposition $\mathcal{T}_{\texttt{PrivHP}}$ output by Algorithm 1 is ε-differentially private for $\sum_{l=0}^{L} \sigma_l = \varepsilon$.

As the proof is standard, it can be found in the full version of the paper.

5.2 Utility and Performance

We begin by introducing some notation. Let $\gamma_l = \max_{\theta \in \{0,1\}^l} \texttt{diam}(\Omega_\theta)$ and $\Gamma_l = \sum_{\theta \in \{0,1\}^l} \texttt{diam}(\Omega_\theta)$. Let $C_l = \langle |\Omega_0|, \ldots, |\Omega_{2^l}| \rangle$ denote the vector of subdomain cardinalities at level l. To help capture the effect of pruning, we use the vector $\texttt{tail}_k^l$ to denote C_l with the top-k cardinalities set to 0. For skewed inputs $||\texttt{tail}_k^l||_1$ is small and can even be 0 for sparse inputs.

Algorithm 1 is general and doesn't prescribe the type of private sketch or the noise distributions of the perturbations. For our concrete results, we use a private Count-min Sketch as the sketching primitive and follow the noise distributions of Lemma 5.1.

Theorem 5.2. *On input $\mathcal{X}$, with sketch dimensions $(w = 2k, j)$ and partition dimensions of pruning level $L_\star$, hierarchy depth L and pruning parameter k, for $\varepsilon = \sum_{l=0}^{L} \sigma_l$, Algorithm 1 produces a partition $\mathcal{T}_{\texttt{PrivHP}}$ that is ε-differentially private and has the following distance from the empirical distribution $\mu_{\mathcal{X}}$ in the expected 1-Wasserstein metric:*

$$\mathbb{E}[W_1(\mu_{\mathcal{X}}, \mathcal{T}_{\texttt{PrivHP}})] = \Delta_{\texttt{noise}} + \Delta_{\texttt{approx}} \tag{4}$$

where

$$\Delta_{\texttt{noise}} = O\left(\frac{1}{n}\left(\sum_{l=0}^{L_\star} \frac{\Gamma_{l-1}}{\sigma_l} + \sum_{l=L_\star+1}^{L} \frac{kj\gamma_{l-1}}{\sigma_l}\right)\right), \quad \Delta_{\texttt{approx}} = O\left(\left(\frac{||\texttt{tail}_k^L||_1}{n} + 2^{-j}\right) \sum_{l=L_\star+1}^{L} \gamma_{l-1}\right)$$

The proof of this result is the content of Section 7. The components allow us to make sense of the bound. The $\Delta_{\texttt{noise}}$ term represents the distance incurred, between $\mu_{\mathcal{X}}$ and $\mathcal{T}_{\texttt{PrivHP}}$, due to noise added for privacy. This noise affects both the counts and the pruning procedure. The $\Delta_{\texttt{approx}}$ term represents the reduction in utility due to approximation. The $||\texttt{tail}_k^L||_1$ term is dependent on the underlying distribution of the input $\mathcal{X}$.

The privacy and accuracy guarantees of Theorems 5.1 and 5.2 hold for any choice of $\{\sigma_l\}$. By optimizing the $\{\sigma_l\}$, we can achieve the best utility for a given level of privacy $\varepsilon = \sum_{l=0}^{L} \sigma_l$.

Lemma 5.3. *With the optimal choice of privacy parameters, on input $\mathcal{X}$ and partition dimensions of $(k, L_\star, L)$, the loss in utility due to noise perturbations ($\Delta_{\texttt{noise}}$ in (4)) is:*

$$\Delta_{\texttt{noise}} = O\left(\frac{1}{\varepsilon n}\left(\sum_{l=0}^{L_\star} \sqrt{\Gamma_{l-1}} + \sum_{l=L_\star+1}^{L} \sqrt{jk\gamma_{l-1}}\right)^2\right)$$

Due to its similarity to Theorem 11 in He *et al* [15], the proof is relegated to the full version of the paper. Theorem 5.2 applies for any input domain Ω. To make the result more tangible and to demonstrate its applicability, following prior work [5, 15], we apply it (in conjunction with Lemma 5.3) to the hypercube $\Omega = [0, 1]^d$. This leads to the following result.

Corollary 5.4. *When $\Omega = [0,1]^d$ equipped with the l^∞ metric, for pruning parameter k,* PrivHP *can process a stream $\mathcal{X}$ of size n in $M = O(k \log^2(n))$ memory and $O(\log(\varepsilon n))$ update time.* PrivHP *can subsequently output a ε-differentially private synthetic data generator $\mathcal{T}_{\texttt{PrivHP}}$, in $O(M \log n)$ time, such that*

$$\mathbb{E}[W_1(\mu_\mathcal{X}, \mathcal{T}_{\texttt{PrivHP}})] = \begin{cases} O\left(\frac{\log^2(M)}{\varepsilon n} + \frac{||\texttt{tail}_k^{\varepsilon n}||}{Mn}\right) & if\, d = 1 \\ O\left(\frac{M^{(1-\frac{1}{d})}}{\varepsilon n} + \frac{||\texttt{tail}_k^{\varepsilon n}||}{M^{1/d} n}\right) & if\, d \geq 2 \end{cases},$$

The proof is provided in Appendix C. For comparison, the state-of-the-art in the static setting, PMM [15], achieves a utility bound of $O(\log^2(\varepsilon n)/(\varepsilon n))$, for $d = 1$, with a memory allocation of $O(\varepsilon n)$ (See Table 1). Thus, through the hierarchy pruning parameter k, PrivHP provides a smooth interpolation from the optimal static case to a memory bounded environment. The same observation is true for $d = 2$. Further, for sparse or highly skewed inputs, where $||\texttt{tail}_k^{\varepsilon n}||_1$ is small, pruning may even improve the utility bound and be preferable in a non-streaming setting, as fewer nodes in the hierarchy results in less noise being added.

6 Measuring Utility

Before proving Theorem 5.2, we need a method to quantify the distance between $\mu_\mathcal{X}$ and $\mathcal{T}_{\texttt{PrivHP}}$. In the empirical distribution, each point $x \in \mathcal{X}$ carries a unit of probability mass. When a point is abstracted into a set within a partition representing a generator, its probability mass is evenly distributed across the set. This reflects the process where, conditioned on a set being selected by the generator, a synthetic point is uniformly sampled from the set. The total distance this probability mass moves during abstraction is bounded by the diameter of the subdomain. Similarly, modifications to node counts in the decomposition tree result in shifts of probability mass within the generator. Bounding the utility of the generator, therefore, involves constraining the distance these probability masses move as $\mu_\mathcal{X}$ transforms into $\mathcal{T}_{\texttt{PrivHP}}$.

To formalize these bounds, we introduce the notion of a *consistency error* to capture inaccuracies arising from both noise perturbations and frequency approximations. This concept enables a precise analysis of how the consistency step balances these errors across nodes. The consistency step ensures coherence in the hierarchy by adjusting a parent node's count and redistributing it among its child nodes. However, inaccuracies in this redistribution—consistency errors—create a divergence between the empirical distribution and the synthetic data generator. The utility cost of a consistency error depends on both its magnitude and the size of the affected subdomain. Therefore, to bound the utility loss from noise and approximation, we must quantify each consistency error and pinpoint its location within the hierarchy.

6.1 Quantifying a Consistency Error

A consistency error represents the transfer of probability mass from one subdomain to its sibling subdomain, altering the probability distribution encoded by the underlying decomposition. To arrive at a formal expression of a consistency error, we begin by introducing some notation. Let $c_\theta = |\Omega_\theta|$ denote the exact count at node v_θ, let λ_θ denote the noise added to $v_\theta.\texttt{count}$ from privacy perturbations, and let e_θ denote the approximation error added to $v_\theta.\texttt{count}$ due to hashing collisions in the sketch. Lastly, we define $\texttt{ConsErr}(v_\theta)$ as the size of the consistency error incurred at node v_θ.

To help quantify $\texttt{ConsErr}(v_\theta)$, we take an accounting approach, where noisy approximate counts are disaggregated into various components using the notation introduced above. This allows us to identify which part of the adjusted consistent counts constitutes an error. As we do not want to double count a consistency error, $\texttt{ConsErr}(v_\theta)$ does not include consistency errors that occur at

ancestor nodes and are, subsequently, inherited in the count at v_θ. Therefore, the size of $\texttt{ConsErr}(v_\theta)$ is solely influenced by the noise and approximation errors in its two child nodes.

A consistency error is dependent on the method employed for consistency. The method we adopt (Algorithm 3) has the following form.

$$\Lambda = v_{\theta 0}.\texttt{count} + v_{\theta 1}.\texttt{count} - v_\theta.\texttt{count}. \tag{5}$$

$$v_{\theta 0}.\texttt{count} \leftarrow v_{\theta 0}.\texttt{count} - \Lambda/2 \quad \text{and} \quad v_{\theta 1}.\texttt{count} \leftarrow v_{\theta 1}.\texttt{count} - \Lambda/2. \tag{6}$$

Note that Algorithm 3 also contains two correction steps for when (6) might violate consistency. We will address these correction steps, and how they affect ConsErr, at a later point.

For clarity, let $\texttt{count}^{\text{before}}$ refer to a node count before consistency is applied and $\texttt{count}^{\text{after}}$ refer to a node count after it is made consistent. Prior to consistency, the count in a child node $v_{\theta 0}$ has the following form:

$$v_{\theta 0}.\texttt{count}^{\text{before}} = c_{\theta 0} + \lambda_{\theta 0} + e_{\theta 0}.$$

Note that $e_{\theta 0} = 0$ if $\texttt{level}(v_{\theta 0}) \leq L_\star$, as no sketches are used. When consistency is enforced at v_θ, consistency has already been applied at the parent of v_θ. Therefore, the following equality already holds:

$$v_\theta.\texttt{count}^{\text{after}} = c_\theta + \texttt{TotErr}_\theta, \tag{7}$$

where $\texttt{TotErr}_\theta$ accumulates consistency errors inherited at v_θ from all its ancestors. To enforce consistency between child nodes, an adjustment variable (See (5)) is calculated:

$$\begin{aligned}\Lambda &= v_{\theta 0}.\texttt{count}^{\text{before}} + v_{\theta 1}.\texttt{count}^{\text{before}} - v_\theta.\texttt{count}^{\text{after}} \\ &= c_{\theta 0} + \lambda_{\theta 0} + e_{\theta 0} + c_{\theta 1} + \lambda_{\theta 1} + e_{\theta 1} - c_\theta - \texttt{TotErr}_\theta \\ &= \lambda_{\theta 0} + e_{\theta 0} + \lambda_{\theta 1} + e_{\theta 1} - \texttt{TotErr}_\theta\end{aligned}$$

Focusing on the left child $v_{\theta 0}$, the size of ConsErr can be inferred by calculating $v_{\theta 0}.\texttt{count}^{\text{after}}$ (See (6)).

$$\begin{aligned}v_{\theta 0}.\texttt{count}^{\text{after}} &= v_{\theta 0}.\texttt{count}^{\text{before}} - \Lambda/2 \\ &= c_{\theta 0} + \lambda_{\theta 0} + e_{\theta 0} - (\lambda_{\theta 0} + e_{\theta 0} + \lambda_{\theta 1} + e_{\theta 1} - \texttt{TotErr}_\theta)/2 \\ &= c_{\theta 0} + (\lambda_{\theta 0} - \lambda_{\theta 1} + e_{\theta 0} - e_{\theta 1})/2 + \texttt{TotErr}_\theta/2\end{aligned} \tag{8}$$

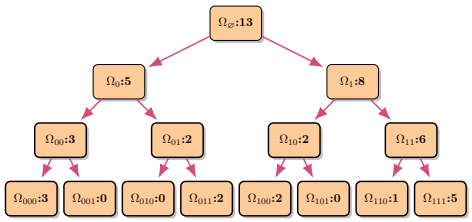
As $\texttt{TotErr}_\theta$ refers to points already counted as errors (which we do not wish to count twice), it follows that

$$\texttt{ConsErr}(v_\theta) = |(\lambda_{\theta 0} - \lambda_{\theta 1} + e_{\theta 0} - e_{\theta 1})/2|. \tag{9}$$

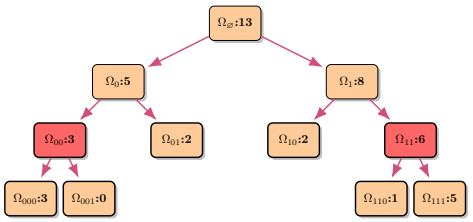
Therefore, a consistency error occurs when there is a difference in the errors between sibling nodes. This difference is evenly split between the subdomains. For greater clarity, a concrete example of a consistency error is provided in Appendix B. With (9) in place, we can now provide a bound for $\mathrm{E}[\texttt{ConsErr}(v_\theta)]$. The proof of the following Lemma is in Appendix D.

Lemma 6.1. *With consistency applied according to Algorithm 3, on sketch parameters* $2w$ *and* j *and the noise distribution from Equation 3, then for all internal nodes* $v_\theta \in \mathcal{T}_{\texttt{PrivHP}}$*:*

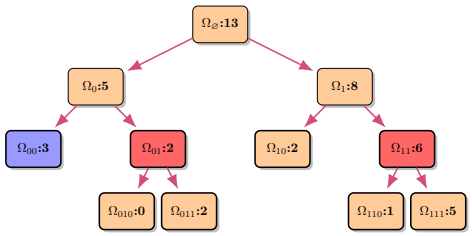
$$\mathbb{E}[\texttt{ConsErr}(v_\theta)] \leq \begin{cases} 2\sqrt{2}\sigma_{l+1}^{-1} & \texttt{level}(v_\theta) < L_\star \\ 2\sqrt{2}\sigma_{l+1}^{-1} \cdot j + \frac{||\text{tail}_w^{l+1}||}{w} + 2^{-j+1}n & \text{Otherwise} \end{cases}$$



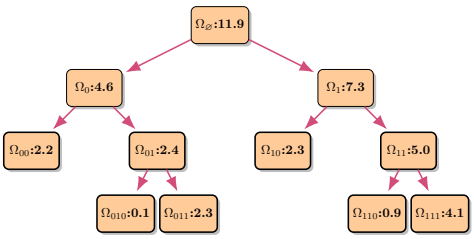
(a) Example of $\mathcal{T}_{\mathcal{X}}$; a complete binary hierarchical decomposition of depth L.



(b) Example of $\mathcal{T}_{\texttt{exact}}$. The distance between $\mu_{\mathcal{X}}$ and $\mathcal{T}_{\texttt{exact}}$ is bound in Lemma 7.1.



(c) Example of $\mathcal{T}_{\texttt{approx}}$. The distance between $\mathcal{T}_{\texttt{exact}}$ and $\mathcal{T}_{\texttt{approx}}$ is bound in Lemma 7.2.



(d) Example of $\mathcal{T}_{\texttt{PrivHP}}$. The distance between $\mathcal{T}_{\texttt{PrivHP}}$ and $\mathcal{T}_{\texttt{approx}}$ is bound in Lemma 7.3.

Fig. 3. Illustration of the proof pipeline for Theorem 5.2. $k = 2$, $L_\star = 2$, $L = 3$.

7 Proof of Theorem 5.2

We break down the proof into a series of steps that are *equivalent* to Algorithm 1. $\mathcal{T}_{\texttt{PrivHP}}$ can be constructed from $\mathcal{X}$ using the following steps:

`Step (1)` Construct a partition tree $\mathcal{T}_{\mathcal{X}}$ that summarizes $\mathcal{X}$ using *exact* counts and a *complete* binary hierarchical partition of depth L (Figure 3a). Then conduct *exact* pruning on $\mathcal{T}_{\mathcal{X}}$ by branching at the nodes with the exact top-k counts at each level $l \geq L_\star$ to produce $\mathcal{T}_{\texttt{exact}}$. An example of $\mathcal{T}_{\texttt{exact}}$ is provided in Figure 3b.

`Step (2)` Constructing $\mathcal{T}_{\texttt{approx}}$ by adjusting $\mathcal{T}_{\texttt{exact}}$ so that its structure matches $\mathcal{T}_{\texttt{PrivHP}}$. $\mathcal{T}_{\texttt{approx}}$ captures the effect of approximate pruning. Note that $\mathcal{T}_{\texttt{approx}}$ still has exact counts. An example of $\mathcal{T}_{\texttt{approx}}$ is provided in Figure 3c.

`Step (3)` Add the privacy noise and approximation errors to the exact counts in $\mathcal{T}_{\texttt{approx}}$ and apply the consistency step to produce $\mathcal{T}_{\texttt{PrivHP}}$. An example of $\mathcal{T}_{\texttt{PrivHP}}$ is provided in Figure 3d.

Figure 3 illustrates the sequence of trees produced by this process. Note that these steps are equivalent to Algorithm 1. However, they are not the same and are only introduced for analytic purposes. By the triangle inequality, it follows that, to bound $\mathbb{E}[W_1(\mu_{\mathcal{X}}, \mathcal{T}_{\texttt{PrivHP}})]$, it suffices to bound the distance between each pair of trees in the sequence. We now bound the cost of each step separately, beginning with `Step (1)`.

At `Step (1)`, exact pruning has the effect of merging sparse sibling leaf nodes into larger subdomains. This reduces the granularity of the partition and, thus, its utility. The exact pruning step allows us to quantify the impact of the underlying data distribution on the utility of the data sampler. For example, distributions that are highly skewed will maintain a majority of their points in top-k nodes. Therefore, pruning will have a smaller impact on the utility of the sampler under high skew. This loss in utility is bounded by the following result.

Lemma 7.1. $\mathbb{E}W_1(\mu_{\mathcal{X}}, \mathcal{T}_{\texttt{exact}}) \leq \frac{1}{n}||\texttt{tail}_k^L||_1 \cdot \sum_{l=L_\star+1}^{L-1} \gamma_l$

Proof. Given the complete binary decomposition of $\mathcal{X}$, the procedure to construct $\mathcal{T}_{\texttt{exact}}$ iterates from level $L_\star + 1$ to the bottom of the tree, selecting to keep the branches with the Top-k counts. Let $\theta_l^{(i)}$ name the candidate node with the i^{th} largest exact cardinality at level l. If a node v_θ is pruned, then each point $x \in \Omega_\theta \cap \mathcal{X}$ is abstracted into subdomain Ω_θ. This is equivalent to redistributing the probability mass of each point in $x \in \Omega_\theta \cap \mathcal{X}$ by moving it a distance of at most $\texttt{diam}(\Omega_\theta)$. As there are $2k$ candidates at each level of pruning, it follows that

$$W_1(\mu_\mathcal{X}, \mathcal{T}_{\texttt{exact}}) \leq \frac{1}{n} \sum_{l=L_\star+1}^{L-1} \sum_{x=k+1}^{2k} |\Omega_{\theta_{(x)}}| \cdot \texttt{diam}(\Omega_{\theta_{(x)}}) \leq \frac{1}{n} \sum_{l=L_\star+1}^{L-1} ||\texttt{tail}_k^l||_1 \gamma_l \leq \frac{||\texttt{tail}_k^L||_1}{n} \cdot \sum_{l=L_\star+1}^{L-1} \gamma_l$$

The final inequality is based on the observation that $||\texttt{tail}_k^{l-1}||_1 \leq ||\texttt{tail}_k^l||_1$, as the subdomains that are top-k in level $l-1$ are split into $2k$ subdomains at level l, which now compete for top-k membership. □

The remaining steps handle the consistency errors. When pruning according to the (noisy and approximate) consistent counts, each consistency error propagates down the tree and affects future pruning decisions. The main challenge in the proof of the bound for `Step (2)` involves demonstrating that the influence of each consistency error *decays* as it propagates down the tree. The following result bounds this utility loss.

Lemma 7.2. $\mathbb{E}W_1(\mathcal{T}_{\texttt{exact}}, \mathcal{T}_{\texttt{approx}}) \lesssim \frac{4}{n} \left(\sum_{l=1}^{L_\star} 2\sqrt{2} \frac{\Gamma_{l-1}}{\sigma_l} + \sum_{l=L_\star+1}^{L} (2\sqrt{2} \frac{jk}{\sigma_l} + ||\texttt{tail}_k^l||_1 + \frac{n}{2^{j-1}}) \gamma_{l-1} \right)$

Proof. Set sketch width $w = 2k$. At `Step (2)`, we adjust the tree structure to reflect the impact of noisy approximate pruning (Figure 3c). Nodes can "jump" into the top-k due to the influence of consistency errors. We refer to the *size* of a jump as the difference in exact counts between the jumping node and the true top-k node it displaces. A jump of size c occurs, for some $\theta_1, \theta_2 \in \{0,1\}^l$, with $l > L_\star$, under the following conditions:

- $v_{\theta_1}.\texttt{count} > v_{\theta_2}.\texttt{count}$;
- $|\Omega_{\theta_1}| + c = |\Omega_{\theta_2}|$ for integer constant $c > 0$;
- and, when v_{θ_2} is in the exact top-k and v_{θ_1} is *not* in the exact top-k.

The *cost* of each jump is at most $c \cdot \gamma_l$, representing c additional points that are abstracted into a subdomain at level l. Therefore, the distance between the sampling distributions defined by $\mathcal{T}_{\texttt{exact}}$ and $\mathcal{T}_{\texttt{approx}}$ is determined by the size of the jumps that occur due to approximate pruning. Further, the size of the jump is bound by the total number of consistency errors that are inherited from ancestors at that node.

A consistency error can cascade through hot nodes, affecting pruning decisions across multiple levels. To formally capture the notion of a specific consistency error propagating down the hierarchy, let $\texttt{cost}_l(v_\theta)$ represent the utility cost incurred at level l due to the consistency error *originating* at v_θ. For example, for a hot node v_θ, at level $l_\theta = \texttt{level}(v_\theta)$, the error $\texttt{ConsErr}(v_\theta)$ is passed to both candidate subdomains $v_{\theta 0}$ and $v_{\theta 1}$. One subdomain will receive $\texttt{ConsErr}(v_\theta)$ as an addition, allowing it to jump upwards, and the other subdomain will receive $\texttt{ConsErr}(v_\theta)$ as a subtraction, allowing it to jump downwards. Therefore, the utility cost incurred by the consistency error of node v_θ at level $l_\theta + 1$ is

$$\texttt{cost}_{l_\theta+1}(v_\theta) \leq \texttt{ConsErr}(v_\theta) \cdot \gamma_{l_\theta+1} + \texttt{ConsErr}(v_\theta) \cdot \gamma_{l_\theta+1} = 2\texttt{ConsErr}(v_\theta) \cdot \gamma_{l_\theta+1}.$$

If these subdomains are both hot, then, due to consistency, $\texttt{ConsErr}(v_\theta)$ is propagated to the next level and is evenly redistributed between the child nodes[2]. Thus, $v_{\theta 00}, v_{\theta 01}, v_{\theta 10}, v_{\theta 11}$ contain (as either an addition or subtraction) $\texttt{ConsErr}(v_\theta)/2$ in their counts. If either subdomain is cold, then

[2]See Equation (8), where errors from the parent node and split evenly in the consistent count of the child node.

its portion of $\texttt{ConsErr}(v_\theta)$ ceases to participate in pruning decisions. Assuming the worst-case, where both subdomains are hot,

$$\texttt{cost}_{l_\theta+2}(v_\theta) \le 4 \cdot \frac{\texttt{ConsErr}(v_\theta)}{2}\gamma_{l_\theta+2} = 2\texttt{ConsErr}(v_\theta)\cdot\gamma_{l_\theta+2} \approx \texttt{ConsErr}(v_\theta)\cdot\gamma_{l_\theta+1},$$

as $\gamma_l \approx 2\gamma_{l-1}$. Repeating this process, assuming, in the worst-case, that all subdomains are hot and continue to propagate $\texttt{ConsErr}(v_\theta)$, it follows that

$$\texttt{cost}_{l_\theta+x}(v_\theta) \le \max\{2^x, k\}\cdot\frac{\texttt{ConsErr}(v_\theta)}{2^{x-1}}\gamma_{l_\theta+x} \le 2\texttt{ConsErr}(v_\theta)\cdot\gamma_{l_\theta+x} \approx 2\texttt{ConsErr}(v_\theta)\cdot\frac{\gamma_{l_\theta+1}}{2^{x-1}},$$

Accumulating these costs, we get

$$\texttt{cost}(v_\theta) = \sum_{x<(L-l_\theta-1)} \texttt{cost}_{l_\theta+x}(v_\theta) \lesssim 4\texttt{ConsErr}(v_\theta)\cdot\gamma_{l_\theta+1}. \tag{10}$$

This applies to nodes v_θ at levels $l_\theta \ge L_\star$. The same analysis extends to nodes v_θ at levels $l_\theta < L_\star$, except that the first level of pruning $\texttt{ConsErr}(v_\theta)$ participates in is $L_\star+1$ (not $l_\theta+1$). Let $\mathcal{H}_l$ denote the hot nodes at level $l \ge L_\star$. Adding everything together, we arrive at the following bound for the distance between $\mathcal{T}_{\texttt{exact}}$ and $\mathcal{T}_{\texttt{approx}}$.

$$\begin{aligned}
W_1(\mathcal{T}_{\texttt{exact}}, \mathcal{T}_{\texttt{approx}})) &= \sum_{l=0}^{L_\star-1}\sum_{\theta:\texttt{level}(\theta)=l}\texttt{cost}(v_\theta) + \sum_{l=L_\star}^{L-1}\sum_{v_\theta\in\mathcal{H}_l}\texttt{cost}(v_\theta)\\
&\lesssim \sum_{l=0}^{L_\star-1}\sum_{\theta:\texttt{level}(\theta)=l} 4\texttt{ConsErr}(v_\theta)\cdot\gamma_{L_\star+1} + \sum_{l=L_\star}^{L-1}\sum_{v_\theta\in\mathcal{H}_l} 4\texttt{ConsErr}(v_\theta)\cdot\gamma_{l+1}\\
&\le \sum_{l=0}^{L_\star-1}\sum_{\theta:\texttt{level}(\theta)=l} 4\texttt{ConsErr}(v_\theta)\cdot\texttt{diam}(\Omega_\theta) + \sum_{l=L_\star}^{L-1}\sum_{v_\theta\in\mathcal{H}_l} 4\texttt{ConsErr}(v_\theta)\cdot\gamma_{l+1}
\end{aligned}$$

Taking expectations, using Lemma 6.1, with sketch parameter $w = 2k$, we arrive at the following.

$$\begin{aligned}
\mathbb{E}[W_1(\mathcal{T}_{\texttt{exact}}, \mathcal{T}_{\texttt{approx}})] &\lesssim \frac{4}{n}\left(\sum_{l=0}^{L_\star-1}\sum_{\theta\in\{0,1\}^l}\mathbb{E}[\texttt{ConsErr}(v_\theta)]\cdot\texttt{diam}(\Omega_\theta) + \sum_{l=L_\star}^{r-1}\sum_{v_\theta\in\mathcal{H}_l}\mathbb{E}[\texttt{ConsErr}(v_\theta)]\cdot\gamma_l\right)\\
&\le \frac{4}{n}\left(\sum_{l=0}^{L_\star-1}\frac{2\sqrt{2}}{\sigma_{l+1}}\sum_{\theta\in\{0,1\}^l}\texttt{diam}(\Omega_\theta) + \sum_{l=L_\star}^{L-1}\gamma_l\sum_{v_\theta\in\mathcal{H}_l}\frac{2\sqrt{2}j}{\sigma_{l+1}} + \frac{||\texttt{tail}_k^{l+1}|| + 2^{-j+1}n}{k}\right)\\
&\le \frac{4}{n}\left(\sum_{l=0}^{L_\star-1}\frac{2\sqrt{2}\Gamma_l}{\sigma_{l+1}} + \sum_{l=L_\star}^{L-1}\left(\frac{2\sqrt{2}jk}{\sigma_l} + ||\texttt{tail}_k^{l+1}||_1 + 2^{-j+1}n\right)\gamma_l\right)\\
&\le \frac{4}{n}\left(\sum_{l=1}^{L_\star}\frac{2\sqrt{2}\Gamma_{l-1}}{\sigma_l} + \sum_{l=L_\star+1}^{L}\left(\frac{2\sqrt{2}jk}{\sigma_l} + ||\texttt{tail}_k^{l}||_1 + 2^{-j+1}n\right)\gamma_{l-1}\right),
\end{aligned} \tag{11}$$

where $\Gamma_l = \sum_{\theta\in\{0,1\}^l}\texttt{diam}(\Omega_\theta)$. □

`Step (2)` bounds the cost of incorrect pruning decisions that occur due to approximations. Note that the bounds for exact pruning (`Step (1)`) and approximate pruning (`Step (2)`) are both expressed in terms of the norm of the tail of the dataset. This because errors in the frequency approximations from the sketches are also bound by the tail norm of their input vectors. This demonstrates that sketches (with width $O(k)$) compose nicely with pruning.

$\mathcal{T}_{\texttt{approx}}$ has the same structure as $\mathcal{T}_{\texttt{PrivHP}}$, but not the same counts. `Step (3)` introduces these (noisy and approximate) counts and, therefore, accounts for the utility loss incurred due to errors in the sampling probabilities.

Lemma 7.3. $\mathbb{E}W_1(\mathcal{T}_{\texttt{approx}}, \mathcal{T}_{\texttt{PrivHP}}) \leq \frac{1}{n}\left(\sum_{l=0}^{L_\star} 2\sqrt{2}\frac{\Gamma_{l-1}}{\sigma_l} + \sum_{l=L_\star+1}^{L}(2\sqrt{2}\frac{jk}{\sigma_l} + ||\texttt{tail}_k^l||_1 + \frac{n}{2^{j-1}})\gamma_{l-1}\right)$

The proof is in the full version of the paper and, similar to Lemma 7.2, entails quantifying the consistency errors and registering where they occur. With a bound on the cost of each step in the proof pipeline, we can proceed with the proof of Theorem 5.2.

Proof of Theorem 5.2. By the triangle inequality, the following holds

$$\mathbb{E}[W_1(\mu_\mathcal{X}, \mathcal{T}_{\texttt{PrivHP}})] \leq \mathbb{E}[W_1(\mu_\mathcal{X}, \mathcal{T}_{\texttt{exact}})] + \mathbb{E}[W_1(\mathcal{T}_{\texttt{exact}}, \mathcal{T}_{\texttt{approx}})] + \mathbb{E}[W_1(\mathcal{T}_{\texttt{approx}}, \mathcal{T}_{\texttt{PrivHP}})]$$

By Lemmas 7.1, 7.2, and 7.3, and the prior observation that $||\texttt{tail}_k^{l-1}||_1 \leq ||\texttt{tail}_k^l||_1$, this evaluates as:

$$\begin{aligned}
&\mathbb{E}[W_1(\mu_\mathcal{X}, \mathcal{T}_{\texttt{PrivHP}})] \\
&\lesssim \frac{1}{n}\left(\sum_{l=0}^{L_\star}\frac{10\sqrt{2}\Gamma_{l-1}}{\sigma_l} + \sum_{l=L_\star+1}^{L}\left(\frac{10\sqrt{2}jk}{\sigma_l} + 5\left(||\texttt{tail}_k^l||_1 + \frac{n}{2^{j-1}}\right)\right)\gamma_{l-1} + ||\texttt{tail}_k^L||\sum_{l=L_\star+1}^{L-1}\gamma_l\right) \\
&\lesssim \frac{10\sqrt{2}}{n}\left(\sum_{l=0}^{L_\star}\frac{\Gamma_{l-1}}{\sigma_l} + \sum_{l=L_\star+1}^{L}\frac{jk\gamma_{l-1}}{\sigma_l}\right) + 6\left(\frac{||\texttt{tail}_k^L||_1}{n} + 2^{-j+1}\right)\sum_{l=L_\star+1}^{L}\gamma_{l-1}
\end{aligned}$$

□

8 Conclusion

In summary, PrivHP offers a novel approach to differentially private synthetic data generation by leveraging a hierarchical decomposition of the input domain within bounded memory constraints. Unlike existing methods, PrivHP provides a principled trade-off between accuracy and space efficiency, balancing hierarchy depth, noise addition, and selective pruning to preserve high-frequency subdomains. Our theoretical analysis establishes rigorous utility bounds, demonstrating that PrivHP achieves competitive accuracy with significantly reduced memory usage compared to state-of-the-art methods. By introducing the pruning parameter k, our approach enables fine-grained control over the trade-off between space and utility, making PrivHP a flexible and scalable solution for private data summarization in resource-constrained environments.

Acknowledgments

This work was supported by CSIRO's Impossible Without You.

References

[1] Daniel Alabi, Omri Ben-Eliezer, and Anamay Chaturvedi. 2022. Bounded space differentially private quantiles. *arXiv preprint arXiv:2201.03380* (2022).

[2] Boaz Barak, Kamalika Chaudhuri, Cynthia Dwork, Satyen Kale, Frank McSherry, and Kunal Talwar. 2007. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 273–282.

[3] Ari Biswas, Graham Cormode, Yaron Kanza, Divesh Srivastava, and Zhengyi Zhou. 2024. Differentially Private Hierarchical Heavy Hitters. *Proceedings of the ACM on Management of Data* 2, 5 (2024), 1–25.

[4] March Boedihardjo, Thomas Strohmer, and Roman Vershynin. 2023. Covariance loss, Szemeredi regularity, and differential privacy. *arXiv preprint arXiv:2301.02705* (2023).

[5] March Boedihardjo, Thomas Strohmer, and Roman Vershynin. 2024. Private measures, random walks, and synthetic data. *Probability theory and related fields* (2024), 1–43.

[6] Yan Chen, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2017. Pegasus: Data-adaptive differentially private stream processing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1375–1388.

[7] Graham Cormode and Donatella Firmani. 2013. On unifying the space of l0-sampling algorithms. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 163–172.

[8] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[9] Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, Entong Shen, and Ting Yu. 2012. Differentially private spatial decompositions. In *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 20–31.

[10] Cynthia Dwork, Aleksandar Nikolov, and Kunal Talwar. 2015. Efficient algorithms for privately releasing marginals via convex relaxations. *Discrete & Computational Geometry* 53 (2015), 650–673.

[11] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[12] Alessandro Epasto, Jieming Mao, Andres Munoz Medina, Vahab Mirrokni, Sergei Vassilvitskii, and Peilin Zhong. 2023. Differentially private continual releases of streaming frequency moment estimations. *arXiv preprint arXiv:2301.05605* (2023).

[13] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2009. Boosting the accuracy of differentially-private histograms through consistency. *arXiv preprint arXiv:0904.0942* (2009).

[14] Yiyun He, Thomas Strohmer, Roman Vershynin, and Yizhe Zhu. 2023. Differentially private low-dimensional representation of high-dimensional data. *arXiv preprint arXiv:2305.17148* (2023).

[15] Yiyun He, Roman Vershynin, and Yizhe Zhu. 2023. Algorithmically effective differentially private synthetic data. In *The Thirty Sixth Annual Conference on Learning Theory*. PMLR, 3941–3968.

[16] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. 2011. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 49–58.

[17] Girish Kumar, Thomas Strohmer, and Roman Vershynin. 2024. PrivStream: An Algorithm for Streaming Differentially Private Data. *arXiv preprint arXiv:2401.14577* (2024).

[18] Christian Janos Lebeda and Jakub Tetek. 2023. Better differentially private approximate histograms and heavy hitters using the Misra-Gries sketch. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 79–88.

[19] Terrance Liu, Giuseppe Vietri, and Steven Z Wu. 2021. Iterative methods for private synthetic data: Unifying framework and new methods. *Advances in Neural Information Processing Systems* 34 (2021), 690–702.

[20] Laurent Meunier, Blaise J Delattre, Alexandre Araujo, and Alexandre Allauzen. 2022. A dynamical system perspective for lipschitz neural networks. In *International Conference on Machine Learning*. PMLR, 15484–15500.

[21] Morteza Monemizadeh and David P Woodruff. 2010. 1-pass relative error lp-samplers with applications. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 1143–1160.

[22] Rasmus Pagh and Mikkel Thorup. 2022. Improved utility analysis of private countsketch. *Advances in Neural Information Processing Systems* 35 (2022), 25631–25643.

[23] Victor Perrier, Hassan Jameel Asghar, and Dali Kaafar. 2018. Private continual release of real-valued data streams. *arXiv preprint arXiv:1811.03197* (2018).

[24] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Understanding hierarchical methods for differentially private histograms. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1954–1965.

[25] Justin Thaler, Jonathan Ullman, and Salil Vadhan. 2012. Faster algorithms for privately releasing marginals. In *International Colloquium on Automata, Languages, and Programming*. Springer, 810–821.

[26] Jonathan Ullman and Salil Vadhan. 2011. PCPs and the hardness of generating private synthetic data. In *Theory of Cryptography Conference*. Springer, 400–416.

[27] Giuseppe Vietri, Cedric Archambeau, Sergul Aydore, William Brown, Michael Kearns, Aaron Roth, Ankit Siva, Shuai Tang, and Steven Z Wu. 2022. Private synthetic data for multitask learning and marginal queries. *Advances in Neural Information Processing Systems* 35 (2022), 18282–18295.

[28] Ulrike von Luxburg and Olivier Bousquet. 2004. Distance-Based Classification with Lipschitz Functions. *J. Mach. Learn. Res.* 5, Jun (2004), 669–695.

[29] Tianhao Wang, Joann Qiongna Chen, Zhikun Zhang, Dong Su, Yueqiang Cheng, Zhou Li, Ninghui Li, and Somesh Jha. 2021. Continuous release of data streams under both centralized and local differential privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1237–1253.

[30] Ziteng Wang, Chi Jin, Kai Fan, Jiaqi Zhang, Junliang Huang, Yiqiao Zhong, and Liwei Wang. 2016. Differentially private data releasing for smooth queries. *Journal of Machine Learning Research* 17, 51 (2016), 1–42.

[31] Yan Yan, Xin Gao, Adnan Mahmood, Tao Feng, and Pengshou Xie. 2020. Differential private spatial decomposition and location publishing based on unbalanced quadtree partition algorithm. *IEEE Access* 8 (2020), 104775–104787.
[32] Jun Zhang, Xiaokui Xiao, and Xing Xie. 2016. Privtree: A differentially private algorithm for hierarchical decompositions. In *Proceedings of the 2016 international conference on management of data*. 155–170.
[33] Fuheng Zhao, Dan Qiao, Rachel Redberg, Divyakant Agrawal, Amr El Abbadi, and Yu-Xiang Wang. 2022. Differentially private linear sketches: Efficient implementations and applications. *Advances in Neural Information Processing Systems* 35 (2022), 12691–12704.

A Proof of Lemma 3.3

We assume that the hash functions are fully random.

Lemma 3.3. *For input vector v, the estimation error of a Count-min Sketch, with width $2w$ and depth j, satisfies, $\forall x \in v$,*

$$\mathbb{E}[\hat{v}_x - v_x] \leq \frac{||\texttt{tail}_w(v)||_1 + 2^{-j+1}||v||_1}{w}$$

where $\text{tail}_w(v)$ *is the vector v with the w largest coordinates removed.*

Proof. Fix an index x and let Top-$w(v)$ denote the set of coordinates in v with the w highest magnitudes. For a given level $i \in [j]$ we can bound the probability that x does not collide with an index in Top-$w(v)$:

$$\begin{aligned}\Pr[h_i(x) \neq h_i(y), \forall y \in \text{Top-}w(v) \setminus \{x\}] &= \prod_{y \in \text{Top-}w(v)\setminus\{x\}} \Pr[h_i(x) \neq h_i(y)] \\ &= \left(\frac{2w-1}{2w}\right)^w,\end{aligned}$$

as the sketch has width $2w$. Taking the logarithm of both sides we get,

$$\begin{aligned}\ln \Pr[h_i(x) \neq h_i(y), \forall y \in \text{Top-}w(v) \setminus \{x\}] &= w \ln\left(1 - \frac{1}{2w}\right) \\ &> w \cdot \left(-\frac{1}{2w} - \left(\frac{1}{2w}\right)^2\right) = -\frac{1}{2} - \frac{1}{4w},\end{aligned}$$

where the inequality comes from the fact that $\ln(1-z) > -z - z^2$ for $z \in (0, 0.5)$. For $w \geq 1$, we have $-\frac{1}{2} - \frac{1}{4w} > -\ln 2$. Thus,

$$\Pr[h_i(x) \neq h_i(y), \forall y \in \text{Top-}w(v) \setminus \{x\}] > e^{-\ln(2)} = \frac{1}{2}.$$

Therefore, the probability that an index does not collide with a Top-w index is greater than a half. Using a standard argument with Chernoff bounds, across j levels, in *at least one level x* has no collisions with the Top-w indices with probability greater than $1 - 2^{-j}$.

We now look at the probability that some some $y \in \text{Top-}w(v) \setminus \{x\}$ observes a collision with x, conditioned on the event that at least one index in Top-$w(v) \setminus \{x\}$ collides with x. Let $\mathcal{E}_i$ denote the event: $\exists y' \in \text{Top-}w(v) \setminus \{x\} : h_i(x) = h_i(y')$. Then $\forall y \in \text{Top-}w(v) \setminus \{x\}$,

$$\Pr[h_i(x) = h_i(y) | \mathcal{E}_i] = \frac{\Pr[h(i(x) = h_i(y) \cap \mathcal{E}_i]}{\Pr[\mathcal{E}_i]} = \frac{1/(2w)}{(1 - 1/(2w))^w} \approx \frac{1/(2w)}{e^{-1/2}} < \frac{2}{w} \tag{12}$$

Let i^* represent the level where x has the fewest collisions with the Top-w indices and $\hat{v}_x^*$ denote the estimate taken from level i^*. As i^* has the fewest collisions, it holds that, $\forall y \in \text{Top-}w(v) \setminus \{x\}$ and

$i \neq i^*$, $\Pr[h_{i^*}(x) = h_{i^*}(y) | \mathcal{E}_{i^*}] \leq \Pr[h_i(x) = h_i(y) | \mathcal{E}_i]$ and can be bound by (12). It follows that,

$$\begin{aligned}
\mathbb{E}[\hat{v}_x] &\leq \mathbb{E}[\hat{v}^*_x] \\
&\leq v_x + \mathbb{E}\left[\sum_{y \in \text{tail}_w(v)} \mathbb{1}[h_{i^*}(x) = h_{i^*}(y)] v_y\right] + \Pr[\mathcal{E}_{i^*}]\left(\mathbb{E}\left[\sum_{y \in \text{head}_w(v)} \mathbb{1}[h_{i^*}(x) = h_{i^*}(y)] v_y \mid \mathcal{E}_{i^*}\right]\right) \\
&\leq v_x + \sum_{y \in \text{tail}_w(v)} \Pr[h_{i^*}(x) = h_{i^*}(y)] \cdot v_y + 2^{-j} \cdot \sum_{y \in \text{head}_w(v)} \Pr[h_{i^*}(x) = h_{i^*}(y) \mid \mathcal{E}_{i^*}] \cdot v_y \\
&= v_x + \frac{||\texttt{tail}_w(v)||_1}{2w} + 2^{-j} \cdot \frac{2 \cdot \sum_{y \in \text{head}_w(v)} v_y}{w} \\
&\leq v_x + \frac{\sum_{y \in \text{tail}_w(v)} v_y + 2^{-j+1}||v||_1}{w}.
\end{aligned}$$

□

Subtracting v_x from both sides completes the proof.

B ConsErr Example

This example evaluates $\texttt{ConsErr}(v_\theta)$ for the subtree in Figure 4. The cardinalities of the subdomains are $c_\theta = 5, c_{\theta 0} = 3, c_{\theta 1} = 2$. From (7), it follows that $\texttt{TotErr}_\theta = v_\theta.\texttt{count}^{\texttt{after}} - c_\theta = -0.4$. The component errors in the child nodes are $\lambda_{\theta 0} = -0.5, e_{\theta 0} = 1, \lambda_{\theta 1} = -0.3, e_{\theta 1} = 2$. Thus, the child counts prior to consistency are

$$v_{\theta 0}.\texttt{count}^{\texttt{before}} = c_{\theta 0} + \lambda_{\theta 0} + e_{\theta 0} = 3.5 \qquad v_{\theta 1}.\texttt{count}^{\texttt{before}} = c_{\theta 1} + \lambda_{\theta 1} + e_{\theta 1} = 3.7.$$

Using the formula in (9), the size of the ConsErr at v_θ is

$$\texttt{ConsErr}(v_\theta) = |(\lambda_{\theta 0} - \lambda_{\theta 1} + e_{\theta 0} - e_{\theta 1})/2| = 0.6$$

This value can be expressed as a portion of the consistent counts in the child nodes.

$$v_{\theta 0}.\texttt{count}^{\texttt{after}} = v_{\theta 0}.\texttt{count}^{\texttt{before}} - \Lambda/2 = c_{\theta 0} + \frac{\texttt{TotErr}_\theta}{2} - \texttt{ConsErr}(v_\theta) = 2.2$$

$$v_{\theta 1}.\texttt{count}^{\texttt{after}} = v_{\theta 1}.\texttt{count}^{\texttt{before}} - \Lambda/2 = c_{\theta 1} + \frac{\texttt{TotErr}_\theta}{2} + \texttt{ConsErr}(v_\theta) = 2.4$$

Therefore, a ConsErr can be understood as the count transferred from one subdomain to its sibling *after* the subdomain cardinalities have been adjusted by the existing error in the parent node. For example, the total error in $v_{\theta 0}.\texttt{count}$ after consistency is 0.8. However, 0.2 of this error comes from consistency errors at ancestor subdomains. Therefore $\texttt{ConsErr}(v_\theta)$ captures the precise count transferred between $\Omega_{\theta 0}$ and $\Omega_{\theta 1}$ due to local errors.

(a) Before consistency (b) After consistency

Fig. 4. Subtree used for Section B.

C Proof of Corollary 5.4

Before proceeding with the proof, we introduce a useful result related to the hypercube $\Omega = [0,1]^d$, which is the domain of Corollary 5.4. The following Lemma bounds the sum of the hypercube subdomain diameters across the pruned levels.

Lemma C.1. *On input domain $\Omega = [0,1]^d$, $d \in Z^+$, privacy budget $\varepsilon > 0$, and hierarchy depth $L = \log \varepsilon n$ and pruning depth $L_\star \geq \log k$,*

$$\sum_{l=L_\star+1}^{L} \gamma_{l-1} = O\left(2^{-L_\star/d}\right).$$

Proof. Let $\Omega = [0,1]^d$ equipped with the l_∞ metric. The natural hierarchical binary decomposition of $[0,1]^d$ (cut through the middle along a coordinate hyperplane) makes subintervals of length $\texttt{diam}(\Omega_\theta) = \gamma_l \asymp 2^{-l/d}$, for $\theta \in \{0,1\}^l$. Note that $\sum_{l=L_\star+1}^{L} \gamma_{l-1} = \sum_{l=L_\star}^{\log(\varepsilon n)-1} 2^{-l/d}$ is a finite geometric series with common ratio $\rho = 2^{-1/d}$. Therefore, we can rewrite the sum using the following formula.

$$\sum_{l=a}^{b} \rho^l = \frac{\rho^a(1-\rho^{b-a+1})}{1-\rho}, \tag{13}$$

where $b = \log(\varepsilon n) - 1$ and $a = L_\star \geq \log k$. Therefore, we get

$$\sum_{l=L_\star}^{\log(\varepsilon n)-1} 2^{-l/d} = 2^{-L_\star/d} \cdot \frac{1-2^{-(\log \varepsilon n - L_\star)/d}}{1-2^{-1/d}} \leq k^{-1/d} \cdot \frac{1-\left(\frac{k}{\varepsilon n}\right)^{1/d}}{1-2^{-1/d}}$$

To evaluate the asymptotics of $\frac{1-\left(\frac{k}{\varepsilon n}\right)^{1/d}}{1-2^{-1/d}}$, we look at its behavior as d and n increase. Clearly, this fraction approaches a constant (for fixed d) as $n \to \infty$. To determine whether this constant depends on d, we need to determine the behavior of the fraction when $d \to \infty$. Both the numerator and the denominator approach 0 as d increases. Therefore, we apply L'Hopital's rule to find the limit. Differentiating the numerator, we get

$$\frac{d}{dd}\left(1-\left(\frac{k}{\varepsilon n}\right)^{1/d}\right) = \frac{\ln\left(\frac{k}{\varepsilon n}\right) \cdot \left(\frac{k}{\varepsilon n}\right)^{1/d}}{d^2}. \tag{14}$$

Differentiating the denominator, we get

$$\frac{d}{dd}(1-2^{-1/d}) = -\frac{\ln 2 \cdot 2^{-1/d}}{d^2} \tag{15}$$

Combining (14) and (15) with L'Hopital's rule, we get

$$\lim_{d\to\infty} \frac{1-\left(\frac{k}{\varepsilon n}\right)^{1/d}}{1-2^{-1/d}} = \lim_{d\to\infty} -\frac{\ln\left(\frac{k}{\varepsilon n}\right) \cdot \left(\frac{k}{\varepsilon n}\right)^{1/d}}{\ln 2 \cdot 2^{-1/d}} = -\frac{\ln\left(\frac{k}{\varepsilon n}\right)}{\ln 2}$$

This is a constant for fixed n. The fraction approaches a constant as either d or n approaches infinity. Therefore, for all values of d and n, $\frac{1-\left(\frac{k}{\varepsilon n}\right)^{1/d}}{1-2^{-1/d}}$ is bound above by some value $C \geq 0$. □

We are now ready to proceed with the Proof of Corollary 5.4. Recall that Corollary 5.4 is an extension of Theorem 5.2 on the hypercube.

Corollary 5.4. *When $\Omega = [0,1]^d$ equipped with the l^∞ metric, for pruning parameter k,* PrivHP *can process a stream $\mathcal{X}$ of size n in $M = O(k \log^2(n))$ memory and $O(\log(\varepsilon n))$ update time.* PrivHP *can subsequently output a ε-differentially private synthetic data generator $\mathcal{T}_{\texttt{PrivHP}}$, in $O(M \log n)$ time, such that*

$$\mathbb{E}[W_1(\mu_{\mathcal{X}}, \mathcal{T}_{\texttt{PrivHP}})] = \begin{cases} O\left(\frac{\log^2(M)}{\varepsilon n} + \frac{||\texttt{tail}_k^{\varepsilon n}||}{Mn}\right) & if d = 1 \\ O\left(\frac{M^{(1-\frac{1}{d})}}{\varepsilon n} + \frac{||\texttt{tail}_k^{\varepsilon n}||}{M^{1/d} n}\right) & if d \geq 2 \end{cases},$$

Proof. We set the sketch depth $j = \lceil \log n \rceil$ and the hierarchy depth $L = \log \varepsilon n$. Therefore, with sketch width $w = 2k$, each sketch occupies $O(k \log n)$ words of memory. As there are at most $L = O(\log \varepsilon n)$ sketches, the memory occupied by the sketches is $O(k \log^2 n)$. If we set $L_\star = O(\log M) = O(\log(k \log^2 n))$, then the memory occupied by the tree of exact counts is also $O(k \log^2 n)$. Therefore, the total memory requirement of PrivHP is $M = O(k \log^2 n)$.

Extending Theorem 5.2 to $\Omega = [0,1]^d$, we proceed by bounding the noise, approximation and resolution terms separately. Starting with the approximation term. Utilizing Lemma C.1 for our choice of $L_\star$, For a sufficiently large constant $C_1 \geq 0$:

$$\Delta_{\texttt{approx}} \leq C_1 \left(\frac{||\texttt{tail}_k^L||}{n} + 2^{-j}\right) \sum_{l=L_\star+1}^{L} \gamma_{l-1} = C_1 \left(\frac{||\texttt{tail}_k^L||}{n} + \frac{1}{n}\right) O(2^{-L_\star/d}) = O\left(\frac{||\texttt{tail}_k^L||}{M^{1/d} n}\right) \quad (16)$$

We now look at the noise term separately for $d = 1$ and $d \geq 2$. For $\Omega = [0,1]$ equipped with the l_∞ metric, the natural hierarchical binary decomposition of $[0,1]$ makes sub-intervals of length $\texttt{diam}(\Omega_\theta) = \gamma_l = 2^{-l}$ for $\theta \in \{0,1\}^l$. Therefore, $\Gamma_l = 1$. The following holds for some constant $C_2 \geq 0$.

$$\begin{aligned} \Delta_{\texttt{noise}}^{d=1} \leq \frac{C_2}{\varepsilon n} \left(\sum_{l=0}^{L_\star} \sqrt{\Gamma_{l-1}} + \sum_{l=L_\star+1}^{L} \sqrt{k j \gamma_{l-1}}\right)^2 &= \frac{C_2}{\varepsilon n} \left(\sum_{l=0}^{L_\star} 1 + (k \log n)^{1/2} \sum_{l=L_\star+1}^{\log(\varepsilon n)} 2^{-(L_\star-1)/2}\right)^2 \\ &= \frac{C_2}{\varepsilon n} \left(L_\star + (k \log n)^{1/2} O(2^{-L_\star/2})\right)^2 \\ &= \frac{C_2}{\varepsilon n} \left(\log M + O\left(\frac{(k \log n)^{1/2}}{(k \log^2 n)^{1/2}}\right)\right)^2 \\ &= O(\log^2(M)/(\varepsilon n)) \end{aligned} \quad (17)$$

Combining (17) and (16), for $\Omega = [0,1]$, we get:

$$\mathbb{E}[W_1(\mu_{\mathcal{X}}, \mathcal{T}_{\texttt{PrivHP}})] = \Delta_{\texttt{noise}} + \Delta_{\texttt{approx}} = O\left(\frac{\log^2(M)}{\varepsilon n} + \frac{||\texttt{tail}_k^{\varepsilon n}||}{Mn}\right)$$

For $\Omega = [0,1]^d$, the natural hierarchical binary decomposition of $[0,1]^d$ makes subintervals of length $\texttt{diam}(\Omega_\theta) = \gamma_l \asymp 2^{-l/d}$, for $\theta \in \{0,1\}^l$. Therefore, $\Gamma_l = 2^l \cdot 2^{-l/d} = 2^{(1-1/d)l}$. We follow the same procedure as above, bounding each term in Theorem 5.2 separately. By Lemma 5.3,

$$\begin{aligned} \Delta_{\texttt{noise}} \leq \frac{C_1}{\varepsilon n} \left(\sum_{l=0}^{L_\star} \sqrt{\Gamma_{l-1}} + \sum_{l=L_\star+1}^{L} \sqrt{k j \gamma_{l-1}}\right)^2 &= \frac{C_1}{\varepsilon n} \left(\sum_{l=0}^{L_\star} \sqrt{2^{(1-1/d)l}} + (k \log n)^{1/2} \sum_{l=L_\star+1}^{L} 2^{-l/(2d)}\right)^2 \\ &= \frac{C_1}{\varepsilon n} \left(O\left(2^{\frac{1}{2}(1-\frac{1}{d})L_\star}\right) + (k \log n)^{1/2} O(2^{-L_\star/(2d)})\right)^2 \\ &= O\left(\frac{M^{(1-\frac{1}{d})/2}}{\varepsilon n} + \frac{(k \log n)^{1/2}}{\varepsilon n (k \log^2 n)^{1/(2d)}}\right)^2 \end{aligned}$$

$$= O\left(\frac{M^{(1-\frac{1}{d})}}{\varepsilon n}\right) \tag{18}$$

The second line comes from Lemma C.1, by setting the dimension to $2d$. Combining (18), (16) and (??), for $\Omega = [0,1]^d$, we get

$$\mathbb{E}[W_1(\mu_\mathcal{X}, \mathcal{T}_{\texttt{PrivHP}})] = \Delta_{\texttt{noise}} + \Delta_{\texttt{approx}} = O\left(\frac{M^{(1-\frac{1}{d})}}{\varepsilon n} + \frac{||\texttt{tail}_k^{\varepsilon n}||}{M^{1/d} n}\right)$$

For the time complexity, at each update, $x \in \mathcal{X}$ performs a root to leaf traversal, updating the counter for each node. Both exact counters are updated in constant time and approximate counters are updated in $O(\log n)$. Therefore the cost of an update is $O(L \cdot \log n) = O(\log^2 n)$.

The partition tree is built one level at a time. The noisy frequency estimates for each node can each be retrieved in $O(j) = O(\log n)$ time. At the first level of pruning $L_\star$, $O(2^{L_\star}) = O(M)$ candidates are retrieved in $O(M \log n)$ time. Then $O(M)$ consistency steps are performed, each in constant time. To retrieve the top-k, they are sorted in $O(M \log M) = o(M \log n)$ time.

The remaining levels $O(L - L_\star) = O(\log n)$ levels output $2k$ candidates. Each frequency estimate is retrieved in $O(\log n)$ and made consistent in constant time. Once the estimates are retrieved, they need to be sorted so that the bottom-k can be pruned. Sorting takes $O(k \log k)$ time. Therefore each remaining level is processed in $O(k \log n)$ time. As there are $O(\log n)$ levels $l > L_\star$, this process requires $O(k \log^2 n)$ time. Therefore, the whole process completes in $O(M \log n)$ time.

□

D Proof of Lemma 6.1

Lemma 6.1. *With consistency applied according to Algorithm 3, on sketch parameters $2w$ and j and the noise distribution from Equation 3, then for all internal nodes $v_\theta \in \mathcal{T}_{\texttt{PrivHP}}$:*

$$\mathbb{E}[\texttt{ConsErr}(v_\theta)] \leq \begin{cases} 2\sqrt{2}\sigma_{l+1}^{-1} & \texttt{level}(v_\theta) < L_\star \\ 2\sqrt{2}\sigma_{l+1}^{-1} \cdot j + \frac{||\texttt{tail}_w^{l+1}||}{w} + 2^{-j+1}n & \text{Otherwise} \end{cases}$$

Proof. We will continue our accounting approach, disaggregating the consistent counts in child nodes into exact counts, the consistency error and errors higher in the hierarchy. Depending on whether error correction is used during consistency, we have three cases to consider:

`Case (1)` No error correction is used;
`Case (2)` `Correction1` is used (Algorithm 3 Line 3);
`Case (3)` `Correction2` is used (Algorithm 3 Line 6).

`Correction1` and `Correction2` have the effect of reducing the amount of error in the node counts. Therefore, they cannot increase the number of misses in a node. We prove this notion formally and consider each case separately.

`Case (1)`. When no error correction is used, $\texttt{ConsErr}(v_\theta)$ is defined in (9). Taking expectation, we get

$$\begin{aligned} \mathbb{E}[\texttt{ConsErr}(v_\theta)] &= \mathbb{E}[|(\lambda_{\theta 0} - \lambda_{\theta 1} + e_{\theta 0} - e_{\theta 1})/2|] \\ &\leq \frac{1}{2}(\mathbb{E}[|\lambda_{\theta 0} - \lambda_{\theta 1}|] + \mathbb{E}[|e_{\theta 0} - e_{\theta 1}|]) \qquad (19) \\ &\leq \mathbb{E}[\max\{|\lambda_{\theta 0}|, |\lambda_{\theta 1}|\}] + \frac{1}{2}(\mathbb{E}[|e_{\theta 0}|] + \mathbb{E}[|e_{\theta 1}|]) \\ &\leq \mathbb{E}[\max\{|\lambda_{\theta 0}|, |\lambda_{\theta 1}|\}] + \frac{||\texttt{tail}_w^{l+1}||}{2w} + 2^{-j+1}n \qquad (20) \end{aligned}$$

The third inequality follows from Lemma 3.3. As the $\lambda_{\theta 0}$ and $\lambda_{\theta 1}$ are independent Laplace variables with noise defined in (3), the following inequality completes the upper bound for case 1.

$$\mathbb{E}[\max\{|\lambda_{\theta 0}|, |\lambda_{\theta 1}|\}] \leq \begin{cases} 2\sqrt{2}\sigma_l^{-1} & l \leq L_\star \\ 2\sqrt{2}\sigma_l^{-1} \cdot j & \text{Otherwise} \end{cases}$$

`Case (2)`. We focus on a correction made to $v_{\theta 0}$. A parallel argument can be made for $v_{\theta 1}$. In `Correction1`, $v_{\theta 0}.\mathtt{count}^{\text{before}}$ is set to 0 if it is negative. As $c_{\theta 0}, e_{\theta 0} \geq 0$, this can only happen if $\lambda_{\theta 0} < 0$. Therefore, under our accounting approach, the correction $v_{\theta 0}.\mathtt{count}^{\text{before}} \leftarrow 0$ is made possible if $\lambda_{\theta 0}$ is changed to some value $|\lambda'_{\theta 0}| \leq |\lambda_{\theta 0}|$. Inserting this value into Inequality (20) has the effect of reducing the bound on the number of misses.

`Case (3)`. As above, we focus on a correction made to $v_{\theta 0}$. A parallel argument can be made for $v_{\theta 1}$. `Correction2` is triggered on node $v_{\theta 0}$, when

$$v_{\theta 0}.\mathtt{count}^{\text{before}} - \Lambda/2 < 0.$$

By (8), this implies that

$$c_{\theta 0} + (\lambda_{\theta 0} - \lambda_{\theta 1})/2 + (e_{\theta 0} - e_{\theta 1})/2 + \mathtt{TotErr}_\theta/2 < 0. \tag{21}$$

Therefore, consistency is violated when any combination of $(\lambda_{\theta 0} - \lambda_{\theta 1})$, $(e_{\theta 0} - e_{\theta 1})$ and $\mathtt{TotErr}_\theta$ are non-positive and sufficiently large. The error correction step entails setting

$$v_{\theta 0}.\mathtt{count}^{\text{after}} \leftarrow 0,$$

thus, *reducing* the amount of error in $v_{\theta 0}.\mathtt{count}^{\text{after}}$. Following our accounting approach, this can be achieved through rescaling $\Lambda/2$ by introducing new error terms $\lambda'_{\theta 0}$ or $e'_{\theta 0}$, with

$$|\lambda'_{\theta 0} - \lambda_{\theta 1}| \leq |\lambda_{\theta 0} - \lambda_{\theta 1}|$$
$$|e'_{\theta 0} - e_{\theta 1}| \leq |e_{\theta 0} - e_{\theta 1}|,$$

such that (21) no longer holds. By inserting these values into (19), we reduce the bound on the consistency error. Therefore, (20) holds in all three cases. □

Received December 2024; revised February 2025; accepted March 2025