PriPL-Tree: Accurate Range Query for Arbitrary Distribution under Local Differential Privacy

Leixia Wang
Renmin University of China
leixiawang@ruc.edu.cn

Qingqing Ye
Hong Kong Polytechnic University
qqing.ye@polyu.edu.hk

Haibo Hu
Hong Kong Polytechnic University
haibo.hu@polyu.edu.hk

Xiaofeng Meng*
Renmin University of China
xfmeng@ruc.edu.cn

ABSTRACT

Answering range queries in the context of Local Differential Privacy (LDP) is a widely studied problem in Online Analytical Processing (OLAP). Existing LDP solutions all assume a uniform data distribution within each domain partition, which may not align with real-world scenarios where data distribution is varied, resulting in inaccurate estimates. To address this problem, we introduce PriPL-Tree, a novel data structure that combines hierarchical tree structures with piecewise linear (PL) functions to answer range queries for arbitrary distributions. PriPL-Tree precisely models the underlying data distribution with a few line segments, leading to more accurate results for range queries. Furthermore, we extend it to multi-dimensional cases with novel data-aware adaptive grids. These grids leverage the insights from marginal distributions obtained through PriPL-Trees to partition the grids adaptively, adapting the density of underlying distributions. Our extensive experiments on both real and synthetic datasets demonstrate the effectiveness and superiority of PriPL-Tree over state-of-the-art solutions in answering range queries across arbitrary data distributions.

1 INTRODUCTION

With increasing personal information collected by third-party entities (a.k.a., data collectors), individual privacy protection is garnering more attention [11, 35, 46, 50]. Local Differential Privacy (LDP) has emerged as a rigorous privacy-preserving standard widely employed in academia and industry [6, 13, 34]. Under LDP, users only need to submit perturbed values, preserving the privacy of their raw data. The data collector collects these noisy values and invests effort in estimating various statistics to support data analysis tasks.

Range queries, as a prevalent query type, have been extensively studied in LDP, where the data collector estimates the frequency of specific ranges within a domain. To support these queries, existing solutions construct hierarchical trees [4, 7, 37, 39] or grids [41, 45] over the whole domain and estimate frequencies of the partitioned subdomains (i.e., nodes in trees or cells in grids). To answer a range query, the frequencies of those nodes or cells covered by the given range will be summed up. When some subdomains are partially covered, the data within them is assumed to be uniformly distributed, so that the corresponding frequency can be estimated based on the overlap proportion with the query range. However, the data we indexed typically exhibits various distributions rather than uniform in reality. It is inevitable to introduce non-uniform errors by existing methods, leading to inaccurate responses.

Figure 1 shows an example of this non-uniform estimation error. Given a distribution depicted as the black curve, Figure 1(a) partitions the domain in a coarse-grained manner, resulting in a large non-uniform error. Figure 1(b) uses finer partitions to reduce the non-uniform error, but incurs significant aggregated LDP noise error due to an increasing number of bins.
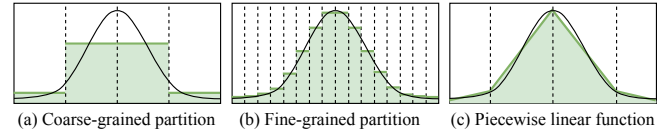

(a) Coarse-grained partition   (b) Fine-grained partition   (c) Piecewise linear function

Figure 1: An Illustration on Non-uniform Errors

To tackle this challenge, we propose an innovative solution that employs a **piecewise linear (PL) function** to model the underlying data distribution instead of relying on a uniform assumption. By partitioning the data domain into several intervals and approximating the data distribution within each interval with a line segment, even complex data distributions can be well-approximated with a few parameters [33]. As shown in Figure 1(c), the PL function accurately approximates the data distribution with a few segments (represented as frequency-slope pairs), which alleviates both non-uniform error and LDP noise error significantly.

Building upon the PL function, we introduce the **Private Piecewise Linear Tree (PriPL-Tree)**. In this tree, each leaf node corresponds to a line segment and stores a frequency-slope pair, while each non-leaf node represents an interval combined from its child

nodes' intervals and stores the associated interval frequency. Compared to traditional hierarchical trees, the PriPL-Tree offers several significant advantages: (1) Within a node with the same interval, it provides a more accurate fit to the underlying data distribution than the uniform assumption. (2) A few segments are sufficient to model the distribution, resulting in fewer leaf nodes and a lower tree height. Height reduction is crucial in LDP as it facilitates allocating users or the privacy budget among fewer tree layers (a necessary step to meet LDP's privacy guarantee), thereby mitigating noise errors in frequency estimation for each node. (3) The number of parameters in the tree depends only on the shape of the data distribution, not the domain size, enabling adaptation to large domains with a more concise structure and more accurate results.

However, constructing the PriPL-Tree under LDP is non-trivial because the data remains invisible to the data collector. To address this, we propose a three-phase approach. First, we allocate a portion of users to estimate the data histogram, gaining a rough glimpse of the underlying data distribution, and use it to fit the PL function. Next, we construct the optimal PriPL-Tree and estimate node frequencies with the remaining users. Finally, we perform postprocessing to refine the frequencies and slopes in the tree, ensuring the non-negativity and consistency of nodes in the tree.

In addition to handling 1-D range queries, we extend PriPL-Tree for multi-dimensional scenarios by incorporating 2-D adaptive grids. These adaptive grids are also data-aware, featuring non-uniform partitions that adapt to the density of the data distribution, and can be constructed utilizing marginal distributions from 1-D PriPL-Trees. By leveraging both 1-D PriPL-Trees and 2-D grids, we can answer λ-D range queries ($\lambda > 1$) using the weighted updating approach [37, 41, 45].

To summarize, our contributions are:

- **Innovative PriPL-Tree**: We design PriPL-Tree, a novel data structure that models the underlying data distribution using a piecewise linear (PL) function instead of relying on a uniform data assumption in LDP. In this way, PriPL-Tree can answer range queries for arbitrary data distributions accurately.
- **Adaptive data-aware Grids**: By leveraging marginal distributions revealed by the PriPL-Trees, we design adaptive grids tailored to the density of underlying data distributions, which serves as the building block for answering multi-dimensional range queries effectively.
- **Extensive Experimental Evaluation**: We conduct comprehensive experiments on both real and synthetic datasets, validating the effectiveness and superiority of our methods. Compared to existing approaches, our method achieves one order of magnitude improvement in accuracy.

In the remainder of this paper, we introduce LDP and analyze existing range query methods in Section 2. Our primary method, PriPL-Tree, is proposed in Section 3, extended with adaptive grids for multi-dimensional cases in Section 4. We evaluate them in Section 5. Finally, we review related works in Section 6 and conclude our paper in Section 7.

2 PRELIMINARIES

In this section, we define the problem and introduce necessary knowledge of LDP and existing methods for range queries in LDP.

2.1 Local Differential Privacy (LDP)

In the context of data collection, LDP provides a mechanism $\mathcal{R}$ that enables users to perturb their data v before sharing it with an untrusted data collector [9, 10, 31]. By ensuring the resulting perturbed data $\mathcal{R}(v)$ satisfies ϵ-LDP, the data collector cannot distinguish a value v from any other possible value v' with high confidence, thus safeguarding the privacy. A higher level of privacy is achieved when a smaller value of ϵ is employed.

Definition 2.1 (ϵ-Local Differential Privacy (ϵ-LDP) [10]). A perturbation mechanism $\mathcal{R}$ satisfies ϵ-LDP ($\epsilon > 0$) iff for any pair of input data $v, v' \in D$ and any output z of $\mathcal{R}$, we have

$$\Pr[\mathcal{R}(v) = z] \le e^\epsilon \Pr[\mathcal{R}(v') = z].$$

We introduce two state-of-the-art LDP mechanisms for fundamental frequency and numerical distribution estimation, respectively, both ensuring ϵ-LDP.

Optimal Unary Encoding Mechanism (OUE) [38] is the state-of-the-art frequency estimation mechanism with three steps: encoding, perturbation, and aggregation. In the encoding step, each user u_i ($i \le N$) encodes his value $v_i \in D$ into a bit vector $\mathbf{B} \in \{0, 1\}^{|D|}$, setting the i-th position to 1 and others to 0. During perturbation, each user perturbs each bit in $\mathbf{B}$ separately. The original bit "1" is retained with probability $p = 1/2$, while the bit "0" is flipped to "1" with probability $q = 1/(e^\epsilon + 1)$. Then, the data collector aggregates all N users' perturbed vectors, counts the number of 1s in the v-th position as n'_v for each v, and calibrates it to an unbiased frequency estimate $\hat{f}_v = (n'_v - Nq)/N(p-q)$, achieving an optimized estimation variance of $\mathrm{Var}(\hat{f}_v) \approx 4e^\epsilon/(N \cdot (e^\epsilon - 1)^2)$, denoted as σ^2.

Square Wave Mechanism (SW) [25] is for numerical distribution estimation, involving perturbation and aggregation steps. In the perturbation step, each user perturbs his value $v_i \in D$ to v'_i within a domain with size $|D| + 2b$, where $b = \left\lfloor \frac{\epsilon e^\epsilon - e^\epsilon + 1}{2e^\epsilon(e^\epsilon - 1 - \epsilon)} \cdot |D| \right\rfloor$. Specifically, with a larger probability $p = e^\epsilon/((2b+1)e^\epsilon + |D| - 1)$, he perturbs v_i to a value v'_i within $|v - v'_i| < b$; with a smaller probability $q = 1/((2b+1)e^\epsilon + |D| - 1)$, he perturbs it to other values. During aggregation, the data collector collects the perturbed data and estimates the distribution using the expectation maximization (EM) algorithm or the EM algorithm with smoothing steps (EMS). SW with EM captures spiky distributions effectively, while SW with EMS provides more accurate estimation by smoothing the LDP noise. We denote these two results as $\hat{\mathbf{F}}^{\mathrm{EM}}$ and $\hat{\mathbf{F}}^{\mathrm{EMS}}$, respectively.

2.2 Problem Definition

Consider N users and each user u_i ($i \le N$) owns a private record containing m private values on attributes $(A_1, A_2, \ldots, A_m)$. Each attribute A_j ($1 \le j \le m$) has a public domain D_j. Each user u_i's record is denoted as $\mathbf{v}_i = (v_i^1, v_i^2, \ldots, v_i^m)$, where $v_i^j \in D_j$ represents the attribute A_j's value for user u_i. For convenience, we assume $D_j = [0, d_j]$ for continuous data and $D_j = \{1, 2, \ldots, d_j\}$ (abbreviated as $[d_j]$) for discrete data. For 1-dimensional (a.k.a., 1-D) data, we abuse v_i to denote the user u_i's value in the default attribute.

The λ-dimensional (a.k.a., λ-D) range query is performed on a set of private attributes $\Phi \subseteq \{A_j | j \le m\}$, where $\lambda = |\Phi| \le m$. Let $[l_j, r_j]$ denote the specified range for the attribute $A_j \in \Phi$. The λ-D range query returns the frequency of records where all queried attribute

Table 1: Notations

Symbols	Description
N	The total number of users
A_j	The j-th attribute
D_j, d_j	The attribute A_j's domain D_j with size d_j
m	The number of private attributes in the data
λ	The number of attributes involved in a range query
n_k	The node n_k in the PriPL-Tree
f_k, β_k	The frequency f_k and the slope β_k in n_k
I_k, s_{k-1}, s_k	The interval I_k of node n_k including $|I_k|$ bucketized values between two breakpoints s_{k-1} and s_k
α	User allocation ratio in phase 1 for PriPL-Tree
σ^2	The variance of OUE with N users and a privacy budget of ϵ

values v_i^j ($j \in \Phi$) are within these specified ranges. Formally,

$$Q\left(\cap_{A_j \in \Phi}[l_j, r_j]\right) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\cap_{A_j \in \Phi}\{l_j \leq v_i^j \leq r_j\}},$$

where $\mathbb{1}$ is an indicator function that outputs 1 if the predicate is true and 0 otherwise.

Our goal is to let the untrusted data collector answer the range query $Q\left(\cap_{j \in \Phi}[l_j, r_j]\right)$ while ensuring individual privacy under ϵ-LDP. Extensive research has been conducted on this problem in the context of LDP, as we reviewed below. The notations used are summarized in Table 1.

2.3 Existing Methods

The hierarchical tree (HT) is the primary data structure for 1-D range queries in LDP. It hierarchically decomposes the entire domain into disjoint sub-domains (a.k.a., intervals), constructing a B-ary tree. Each node in the tree represents an interval and stores an estimated interval frequency. Non-leaf nodes aggregate frequencies of their B child nodes. The data within leaf nodes is assumed to be uniformly distributed. As such, range queries can be answered by summing a few node frequencies (or parts of them) in the tree rather than all individual bins' frequencies within the range, as in a histogram, reducing the accumulated noise error. For example, considering a domain $D = [0, 16]$, we can either uniformly partition it into 16 bins for a histogram or construct a complete binary tree with 16 leaves. Given a range query $Q([0, 5])$, it can be answered by summing the two frequencies of nodes with intervals $[0, 4)$ and $[4, 5]$ in the tree, rather than five frequencies of individual bins $[0, 1), [1, 2), [2, 3), [3, 4)$ and $[4, 5]$ in the histogram. To build this tree under LDP, the privacy budget or users are allocated among layers to estimate nodes' frequencies. To further reduce the error of range queries, a lot of optimization methods have been developed, including Haar transformation of data [4], optimizing the branch number for the tree [4, 39], customizing branch numbers for nodes [37] and merging nodes with low frequencies [7].

Beyond 1-D, the HT can be extended to multi-dimensional cases [7, 39]. However, finely partitioning users or the privacy budget among layers and dimension combinations would increase the noise error. To overcome the curse of dimensionality, grid-based methods are typically employed in λ-D ($\lambda > 1$) range queries. Given m private attributes, they estimate the frequencies of m 1-D grids for individual attributes and $\binom{m}{2}$ 2-D grids for attribute pairs. Users

or privacy budgets are allocated among these grids, where the data in each cell is still assumed to be uniformly distributed. Based on these, a λ-D range query can be estimated from these 1-D and 2-D grids through the maximum entropy [51] or weighted updating [45] algorithms. To reduce the estimation error of range queries, Yang et al. [45] optimized the granularity for both 1-D and 2-D grids, and Wang et al. [41] further employed prefix-sum (PS) cubes.

2.4 Observations and Challenges

Drawing from current research on range queries in LDP, we summarize two key observations that guide our approach and identify a significant challenge. First, we outline the observations:

(1) Tree vs. Grid: Tree-based methods allocate users (or privacy budget) across multiple layers and dimensions, whereas grid-based methods allocate only among dimensions. Considering the significant noise from a few users or a small privacy budget, tree-based methods are preferable for 1-D range queries, while grid-based methods are more suitable for λ-D ($\lambda > 1$) range queries [37].

(2) User Allocation vs. Privacy Budget Allocation: To achieve ϵ-LDP, allocation of users or privacy budget is necessary among the layers of trees and the grids. Generally, user allocation is preferable in the LDP setting as it introduces less noise error than privacy budget allocation [4, 7, 37, 39, 45].

We then present a significant challenge: **unrealistic uniform assumptions**. All existing works decompose the domain uniformly and/or assume uniform data distribution in each decomposed sub-domain [4, 7, 37, 39, 41, 45]. However, real-world applications often involve data following various distributions (e.g., Gaussian, Zipf) rather than being uniformly distributed [22, 42]. This assumption inevitably leads to non-uniform errors and suboptimal estimates.

In this work, we address uniform assumptions on the domain decomposition and data, and correspondingly provide enhanced estimation accuracy for range queries in LDP. In what follows, we first present a solution for 1-D range queries in Section 3 and then extend it to a multi-dimensional setting in Section 4.

3 PRIVATE PIECEWISE LINEAR TREE

In this section, we propose PriPL-Tree, a private piecewise linear tree that combines piecewise linear functions and hierarchical trees to address uniform assumptions for 1-D range queries.

3.1 Design Rationale

The piecewise linear (PL) function is capable of approximating the underlying data distribution with only a few parameters, enabling us to not rely on uniform distribution assumptions. For instance, given a Gaussian distribution in Figure 2(a), we can approximate it using 4 segments with 8 parameters. In this case, the entire domain is divided into 4 intervals, and data in each interval $I_k = [s_{k-1}, s_k)$ ($1 \leq k \leq 4$) is fitted with a linear function defined by two parameters, i.e., β_k (slope) and f_k (sum of frequencies of all points in the interval). The linear expression is given by $y = \beta_k x + b_k$, where $b_k = f_k/|I_k| - \beta_k(|I_k| + 2s_{k-1} - 1)/2$ and $|I_k|$ is the interval size.

To facilitate range query processing, we integrate the PL function with a hierarchical tree structure, proposing the **Private Piecewise Linear Tree (PriPL-Tree)**. Each leaf node represents a segment (corresponding to an interval) of the PL function and stores its slope

β_k and frequency f_k. Each non-leaf node represents an interval and only stores the interval frequency. Like conventional hierarchical trees, the parent node stores the sum of its child nodes' frequencies. We count the layers of the tree starting from 0 at the top.
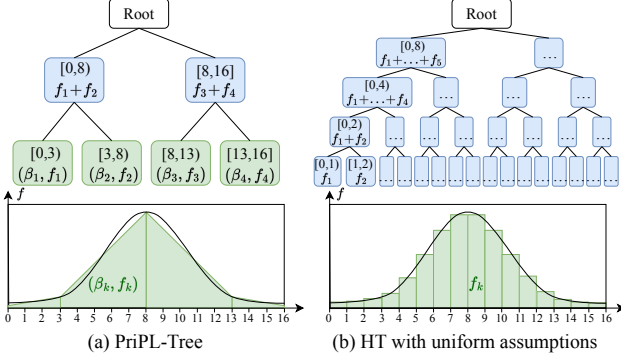

Figure 2: An Example of PriPL-Tree and HT

We provide an example in Figure 2 to illustrate the PriPL-Tree, comparing it to the conventional hierarchical tree (HT) with uniform assumptions. Obviously, within the same interval, the PL function can provide a more accurate approximation of the underlying distribution than uniform assumptions. As such, the PriPL-Tree captures the underlying distribution using significantly fewer leaf nodes (4 in PriPL-Tree vs. 16 in HT) and correspondingly fewer layers (2 in PriPL-Tree vs. 4 in HT). In the context of LDP, fewer layers mean each layer in the tree can be allocated more users, resulting in less noise error due to the law of large numbers. Moreover, the PriPL-Tree construction depends solely on the distribution of the underlying data, as opposed to HT, which relies on the domain size. In HT, modeling data with a large domain size requires a taller tree or a coarser granularity for leaf nodes, increasing noise errors or non-uniform errors. The PriPL-Tree is well-suited for large domain-sized scenarios while reducing both two types of errors.

However, constructing an effective PriPL-Tree in LDP settings is challenging due to the invisible data distribution. To address this, we first employ some users to collect a noisy histogram using LDP mechanisms, gaining insight into the underlying data distribution. We then fit PL functions based on this noisy histogram and use the remaining users to construct the tree. Through post-processing, we further optimize these estimated frequencies and slopes to maintain tree consistency and improve range query accuracy. Following this idea, we propose a three-phase workflow as outlined below and detail the methods for each phase in separate subsections.

3.2 Workflow of PriPL-Tree

The workflow of the PriPL-Tree involves three phases: Private PL Fitting, PriPL-Tree Construction, and PriPL-Tree Refinement, exemplified in Figure 3.

Phase 1: Private Piecewise Linear (PL) Fitting. To gain fundamental insight into the data distribution, we employ a proportion α of users to execute SW protocols with the privacy budget ϵ, collecting a noisy histogram $\hat{\mathbf{F}}^{\mathrm{H}}$ on the bucketized domain $[d]$. Then, we fit

the PL function over this histogram, as presented in Section 3.3. During PL fitting, we address two key issues: interval partitioning and segment fitting. Interval partitioning involves determining the number of intervals K and identifying $K+1$ breakpoints $\{s_0, s_1, ..., s_K\}$. The derived intervals are denoted as $I_k = [s_{k-1}, s_k)$ for $1 \le k < K$ and $I_K = [s_{K-1}, s_K]$ for the last interval. For mapping to the histogram, we can also mark $[s_{k-1}, s_k)$ as $[s_{k-1}, s_k-1]$. Segment fitting focuses on fitting the slope parameter $\beta_k (1 \le k \le K)$ of the line segment for each interval. Although an intercept parameter of the PL function is also derived, we do not record it, only the slope parameter β_k and the estimated interval frequency, i.e., the sum of frequencies of values in each interval, denoted as $\hat{f}_k = \sum_{v \in S_k} \hat{f}_v^{\mathrm{H}} (1 \le k \le K)$. These two parameters can be further optimized using the collected interval frequencies in subsequent phases.

Phase 2: PriPL-Tree Construction. Based on the PL function, we dynamically construct the PriPL-Tree structure in this phase, as detailed in Section 3.4. Each leaf node corresponds to a fitted segment in sequence, e.g., n_1 to interval I_1 and n_2 to interval I_2, as shown in Figure 3 (a) and (b). Each non-leaf node represents an interval encompassing its children and has a non-uniform branch number (i.e., fan-out). This flexible structure is designed to minimize average error in responding to range queries.

Given the PriPL-Tree structure, we allocate the remaining $N \cdot (1-\alpha)$ users to nodes and estimate their frequencies. Because the intervals of nodes along each path from the root to the leaves overlap, each user is randomly allocated to one node per path. As a result, the total number of users along each path is $N \cdot (1-\alpha)$. Each individual user is assigned multiple nodes with non-intersecting intervals that jointly cover the entire domain. Informed of these intervals, users can encode their values into bit vectors to employ the OUE mechanism with privacy budget ϵ for frequency estimation. For example, if a user's value v is covered by nodes $\{n_3, n_7\}$ and he receives the intervals of nodes (n_6, n_3, n_4, n_5), he can encode his value as $(0, 1, 0, 0)$ and apply OUE. By aggregating all users' perturbed values for corresponding nodes, we derive each node's frequency $\bar{f}_k$, forming a preliminary PriPL-Tree, as shown in Figure 3 (b). Each leaf node has two frequencies: $\hat{f}_k$, estimated during private PL fitting in phase 1, and $\bar{f}_k$, estimated by OUE in this phase.

Phase 3: PriPL-Tree Refinement. In the current PriPL-Tree, there are several frequency inconsistencies: (1) the estimated frequency of values or intervals may be beyond the actual range of $[0, 1]$, (2) the frequency of a parent node may differ from the frequency sum of its child nodes, (3) two different frequencies occur at leaf nodes. To address these issues, we propose a post-processing method in Section 3.5, yielding an optimized PriPL tree as shown in Figure 3 (c). It has consistent frequencies $\tilde{\mathbf{F}}$ across all nodes and optimized slopes $\tilde{\beta}_k$ for leaf nodes. By now, a well-estimated PriPL-Tree is ready to respond to range queries.

Response to 1-D Range Query $Q([l, r])$. Given a 1-D range query $Q([l, r])$, the response is obtained by summing the frequencies $\tilde{f}_k$ of nodes n_k that are fully within the range $[l, r]$ but whose parents are not, as well as frequencies from parts of leaf nodes that overlap but are not completely within $[l, r]$. For example, when querying $Q([200, 1024])$ in Figure 3 (c), we aggregate the frequencies $\tilde{f}_2$ of node n_2, $\tilde{f}_7$ of node n_7, and the frequency of the sub-range $[200, 340]$ (i.e., $[200, 339]$) within node n_1. All these nodes and their

(a) Phase 1: Private Piecewise Linear Fitting
(with $N \cdot \alpha$ users)

(b) Phase 2: PriPL-Tree Construction
(with $N \cdot (1 - \alpha)$ users)

(c) Phase 3: PriPL-Tree Refinement
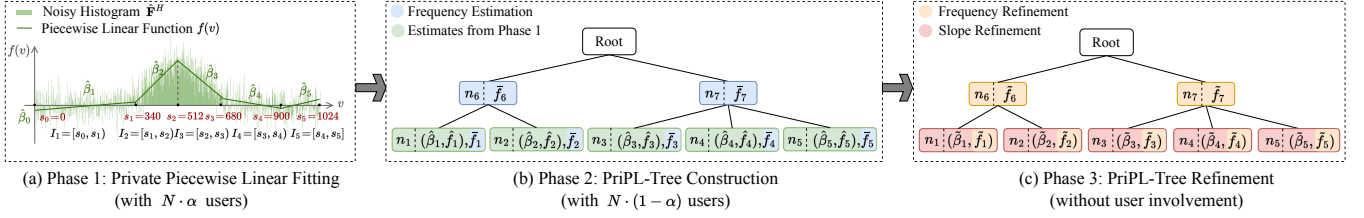(without user involvement)

Figure 3: Workflow of PriPL-Tree

corresponding frequencies can be derived by traversing the PriPL-Tree from top to bottom. Let $[l_{\text{sub}}, r_{\text{sub}}]$ denote the intersecting range of $[l, r]$ with the interval I_k of a leaf node n_k; the frequency of this sub-range $Q([l_{\text{sub}}, r_{\text{sub}}])$ can be computed using Eq. (1). The detailed computation process is shown in Appendix A.1 in [36].

$$Q([l_{\text{sub}}, r_{\text{sub}}]) = (r_{\text{sub}} - l_{\text{sub}} + 1) \cdot \left(\tilde{\beta}_k \left(\tfrac{l_{\text{sub}} + r_{\text{sub}} + 1 - |I_k|}{2} - s_{k-1} \right) + \tfrac{\tilde{f}_k}{|I_k|} \right) \quad (1)$$

3.3 Private PL Fitting

As a fundamental data model, the PL function has been extensively studied in stream compression [2, 12, 21, 26, 44] and learned index [14, 15, 23, 24] applications. In these contexts, the original data distribution is available, and the PL model can be learned using heuristic algorithms with specified error restrictions. However, our task poses a key challenge as we aim to use a noisy histogram to fit an unknown distribution while achieving an optimal error. To address this challenge, we carefully design the following segment fitting and interval partitioning steps to learn a PL model on the data distribution. The pseudocode is provided in Algorithm 1.

3.3.1 Segment Fitting. Let's start with a simple case with K partitioned intervals, and we aim to optimize the PL function by minimizing the squared error between the fitted and the noisy values. To alleviate the impact of LDP noise, we assume the PL function is continuous. This allows us to model the entire noisy histogram as a whole, leveraging all histogram data to fit each line segment, rather than using only a subset of data located in individual intervals, which may be overwhelmed by LDP noise. Moreover, this assumption is practical even for non-continuous distributions, as data around the breakpoints can be approximated by a continuous function with a sharp line connecting two breakpoints. Such approximation would produce only minor errors, especially for bucketized histogram values.

We model the continuous PL function in Eq.(2) and illustrate it with $K = 5$ segments in Figure 3 (a). Each s_k ($0 < k < K$) represents a breakpoint between two segments, while s_0 and s_K mark the domain's endpoints. Let β_0 denote the intercept of the first segment. For each segment in the interval I_k, its slope is denoted by β_k, and its linear expression appears in the k-th row of Eq.(2).

$$f(v) = \begin{cases} \beta_0 + \beta_1(v - s_0), & s_0 \le v < s_1 \\ \beta_0 + \beta_1(s_1 - s_0) + \beta_2(v - s_1), & s_1 \le v < s_2 \\ \dots \\ \beta_0 + \sum_{k=1}^{K-1} \beta_k(s_k - s_{k-1}) + \beta_K(v - s_{K-1}), & s_{K-1} \le v \le s_K \end{cases} \quad (2)$$

Given the noisy histogram with frequency $\hat{f}_v^{\text{H}}$ for $v \in [d]$, our objective is to minimize the squared error between the PL function $f(v)$ and the observed frequencies, i.e., $\min \sum_{v \in [d]} (f(v) - \hat{f}_v^{\text{H}})^2$.

For ease of optimization, we express the problem using matrices. Let $\mathbb{1}$ represent an indicator function, which is 1 when its predicate is met and 0 otherwise. We denote the frequencies in the noisy histogram by the vector $\hat{\mathbf{F}}^{\text{H}} = [\hat{f}_1^{\text{H}} \ \hat{f}_2^{\text{H}} \ \dots \ \hat{f}_d^{\text{H}}]^T$, and the parameters of the PL function by $\mathbf{B} = [\beta_0 \ \beta_1 \ \dots \ \beta_K]^T$. We define two matrices, $\mathbf{X}_{d \times (K+1)}$ and $\mathbf{A}_{d \times (K+1)}$, as described in Eq.(3) and Eq.(4) respectively. In both matrices, the v-th ($v \in [d]$) row corresponds to the expression of $f(v)$ by Eq.(2). For $k > 0$, the k-th column in $\mathbf{X}$ refers to the term $(v - s_{k-1})$, and the k-th column in $\mathbf{A}$ refers to the term $(s_k - s_{k-1})$.

$$\mathbf{X} = \begin{bmatrix} 1 & (1 - s_0)\mathbb{1}_{s_0 \le 1 < s_1} & \cdots & (1 - s_{K-1})\mathbb{1}_{s_{K-1} \le 1 \le s_K} \\ 1 & (2 - s_0)\mathbb{1}_{s_0 \le 2 < s_1} & \cdots & (2 - s_{K-1})\mathbb{1}_{s_{K-1} \le 2 \le s_K} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & (d - s_0)\mathbb{1}_{s_0 \le d < s_1} & \cdots & (d - s_{K-1})\mathbb{1}_{s_{K-1} \le d \le s_K} \end{bmatrix} \quad (3)$$

$$\mathbf{A} = \begin{bmatrix} 0 & (s_1 - s_0)\mathbb{1}_{1 > s_1} & \cdots & (s_{K-1} - s_{K-2})\mathbb{1}_{1 > s_{K-1}} & 0 \\ 0 & (s_1 - s_0)\mathbb{1}_{2 > s_1} & \cdots & (s_{K-1} - s_{K-2})\mathbb{1}_{2 > s_{K-1}} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & (s_1 - s_0)\mathbb{1}_{d > s_1} & \cdots & (s_{K-1} - s_{K-2})\mathbb{1}_{d > s_{K-1}} & 0 \end{bmatrix} \quad (4)$$

By calculating $(\mathbf{X} + \mathbf{A}) \cdot \mathbf{B}$, we derive the PL fitted frequencies for all values in $[d]$, i.e., $[f(1) \ f(2) \ \dots \ f(d)]^T$. Consequently, we reformulate the optimization problem using matrices as follows:

$$\min((\mathbf{X} + \mathbf{A}) \cdot \mathbf{B} - \hat{\mathbf{F}}^{\text{H}})^T \cdot ((\mathbf{X} + \mathbf{A}) \cdot \mathbf{B} - \hat{\mathbf{F}}^{\text{H}}). \quad (5)$$

Let $L(\mathbf{B}) = ((\mathbf{X} + \mathbf{A}) \cdot \mathbf{B} - \hat{\mathbf{F}}^{\text{H}})^T \cdot ((\mathbf{X} + \mathbf{A}) \cdot \mathbf{B} - \hat{\mathbf{F}}^{\text{H}})$ denote the loss function. By setting $\partial L(\mathbf{B}) / \partial \mathbf{B} = 0$, we derive the closed-form solution for $\hat{\mathbf{B}}$, where $\hat{B}_0$ represents the estimated intercept $\hat{\beta}_0$ and $\hat{B}_k$ ($k > 0$) represents the estimated slope $\hat{\beta}_k$ of the k-th segment.

$$\hat{\mathbf{B}} = ((\mathbf{X} + \mathbf{A})^T (\mathbf{X} + \mathbf{A}))^{-1} (\mathbf{X} + \mathbf{A})^T \hat{\mathbf{F}}^{\text{H}}. \quad (6)$$

3.3.2 Interval Partitioning. Building upon segment fitting, we propose a greedy method to search breakpoints one by one, achieving approximately optimal interval partitions. As described in Algorithm 1, during each search (i.e., each iteration in lines 5~12), we traverse all candidate breakpoints s in the search space Θ, fit segments based on it and the existing breakpoints $\mathbf{S}$, and find the best breakpoint s^* that minimizes the residual sum of squares (RSS). The initial search space contains all possible candidates in the whole domain. In subsequent iterations, we select values in the interval I_{k*} as the new search space. This interval I_{k*} has the maximum RSS

Algorithm 1: Private PL Fitting

Input: Noisy histograms $\hat{\mathbf{F}}^{EM}$ and $\hat{\mathbf{F}}^{EMS}$ by SW mechanism.
Output: A PL function with adaptive K segments.

1   Set segment number $K = 1$ and breakpoints $\mathbf{S} = (0, d-1)$;
2   Initialize search space $\Theta = \{s | s \in [d-1]\}$;
3   **for** $i \in \{1, 2\}$ **do**
4   **if** $i == 1$ **then** $\hat{\mathbf{F}}^{H} = \hat{\mathbf{F}}^{EM}$ **else** $\hat{\mathbf{F}}^{H} = \hat{\mathbf{F}}^{EMS}$;
5   **while** *RSS not converges or* $K \leq (K_{max} \times i/2)$ **do**
6       $K = K + 1$;
7   **foreach** *Breakpoint candidate s in* Θ **do**
8   Initialize $\mathbf{X}_{d \times (K+1)}$ and $\mathbf{A}_{d \times (K+1)}$ with $\mathbf{S}$ and s;
9   Calculate PL parameters $\hat{\mathbf{B}}$ with Eq.(6);
10   Calculate $RSS_k = \sum_{v \in I_k} (\hat{f}_v^{H} - f(v))^2$ for each interval S_k, record total $RSS = \sum_{1 \leq k \leq K} RSS_k$;
11   Append s^* to $\mathbf{S}$, where $s^* \in \Theta$ produces the minimal RSS;
12   Set $\Theta = \{s | s \in I_{k^*}\}$, where I_{k^*} has the maximum RSS_{k^*} and its frequency $\hat{f}_k = \sum_{v \in I_k} \hat{f}_v^{EMS} > \sigma\sqrt{(1-\alpha)}$;
13   **return** A PL function with breakponts $\mathbf{S}$, slopes $\hat{\mathbf{B}}$ and frequencies $\hat{\mathbf{F}}$;

Algorithm 2: PriPL-Tree Construction

Input: K segments and $N(1-\alpha)$ users
Output: PriPL-Tree $\mathcal{T}$
// Tree Structure Construction

1   Construct a basic balanced binary tree $\mathcal{T}$;
2   **for** *node* n_k *in postorder traversal* **do**
3   Compute Err for $\mathcal{T}$ with n_k and Err' w/o n_k using Eq.(7);
4   **if** $Err' < Err$ **then** Remove n_k ;
    // User Allocation
5   Assign all available users $U_0 = \{u_1, u_2, \ldots, u_{N(1-\alpha)}\}$ to the root;
6   **for** *node* n_k *in level order traversal* **do**
7   **if** n_k *is root* **then** Set $U'_k = \phi$;
8   **else**
9   Compute h_k, the height of the subtree rooted at n_k;
10   Set U'_k as $\left\lceil \frac{|U_k|}{h_k} \right\rceil$ randomly sampled users from U_k;
11   Allocate U'_k to n_k for frequency estimation;
12   **for** *node* $n_c \in children(n_k)$ **do** Assign $U_c = U_k - U'_k$ to n_c ;
    // Node Frequency Estimation
13   **for** *each user* $u_i \in U_0$ **do**
14   Assign intervals of nodes $\mathcal{N}_i = \{n_k | u_i \in U'_k\}$ to u_i;
15   Collect OUE perturbed vectors with size $|N_i|$;
16   Estimate frequency $\bar{f}_k$ for each node n_k based on OUE;
17   **return** PriPL-Tree $\mathcal{T}$;

---

for its fitted line segment, indicating it requires further splitting for a more accurate fit. Additionally, its frequency $\hat{f}_k$ should be greater than $\sigma\sqrt{(1-\alpha)}$, ensuring that it will not be overwhelmed by OUE noise in node frequency estimation in the next phase. This iteration continues until the maximum number of segments is reached (we set $K_{max} = 32$ in experiments) or RSS converges, i.e., the ratio of total RSS between two consecutive iterations approaches 1, indicating no further gain from increasing segments.

Additionally, we propose two strategies to improve interval partitioning for effectiveness and efficiency. These are briefly described below, with detailed explanations provided in Appendix A in [36].

(1) *Twice Partitioning Strategy for Effectiveness*: To fit both jagged and smooth distributions, we sequentially perform interval partitioning on two distributions, $\hat{\mathbf{F}}^{EM}$ and $\hat{\mathbf{F}}^{EMS}$, as outlined in lines 3~4 of Algorithm 1. These distributions, derived from SW using EM and EMS, respectively, represent an initially calibrated (typically jagged) distribution and a smoothed one with reduced noise [25]. After this process, we derive the necessary partitions to depict both types of distributions. Notably, these two estimations from SW require only one perturbation per user, not increasing the privacy budget.

(2) *Search Acceleration Strategy for Efficiency*: To accelerate the search process, we propose a multi-granular search strategy instead of traversing all possible breakpoints at each search in line 7 in Algorithm 1. Given the granularity factor ϕ, which limits the maximum number of candidate breakpoints during each search, we initially explore the space Θ using a step of $\lceil |\Theta|/\phi \rceil$ to identify an optimal breakpoint s^*. Subsequently, we narrow the search space to $[s^* - \lceil |\Theta|/\phi \rceil, s^* + \lceil |\Theta|/\phi \rceil]$ and search it with a finer step of $\lceil |\Theta|/\phi^2 \rceil$. We repeat this process until the step size reduces to 1. A relatively smaller ϕ ($\phi < d$) accelerates the search while increasing the probability of encountering local optima. Since breakpoints inherently represent different local optima partitioning the domain, ϕ primarily influences the order of breakpoint discovery rather than the final interval partitions. For brevity, this strategy is not included in Algorithm 1, but it can replace line 7 of it.

3.4 PriPL-Tree Construction

Based on the derived partitioned intervals, we construct the PriPL-Tree, focusing on user allocation and tree structure construction and providing the pseudocode in Algorithm 2.

3.4.1 User Allocation. Given a potentially unbalanced PriPL-Tree $\mathcal{T}$, we explore user allocation strategies. Nodes along each path from the root to the leaves have overlapping intervals and related frequencies, prompting us to allocate users to nodes along paths [37] rather than by layers [4, 7, 39]. The allocation process is detailed in lines 5~12 of Algorithm 2. Initially, all unallocated users are assigned to the root (line 5). Because the root has a constant frequency of 1 and requires no estimation, it is allocated no users (line 7) and just passes the user set to its children. For each non-root node n_k that we traversed in level order (line 6), it has inherited the unallocated user set U_k from its parent (line 12), uniformly samples $1/h_k$ of these users for itself, marked as U'_k (lines 10~11), and passes the remaining users $U_k - U'_k$ to its children (line 12). Here, h_k represents the height of the subtree rooted at n_k, ensuring uniform allocation along the longest path. During this process, all child nodes of n_k will receive the same to-be-allocated user group $U_k - U'_k$ since they do not overlap in their intervals.

An example of user allocation is shown in Figure 4, where colored rectangles above each node represent the randomly assigned users. Along each path, such as "$n_0 - n_7 - n_8 - n_3$" in Figure 4 (a), the total number of users is N'. For each user, like the one represented by the yellow rectangle, he will participate in frequency estimations for multiple nodes, such as $\{n_6, n_3, n_4, n_5\}$. The intervals of these nodes do not intersect and collectively cover the entire domain.
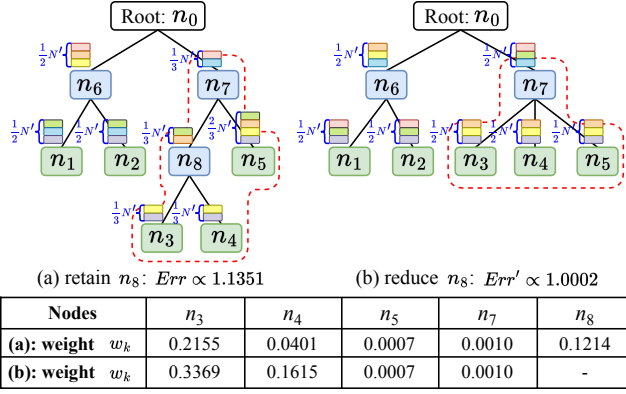
Nodes	n_3	n_4	n_5	n_7	n_8
(a): weight w_k	0.2155	0.0401	0.0007	0.0010	0.1214
(b): weight w_k	0.3369	0.1615	0.0007	0.0010	-

Figure 4: An Example of Tree Construction ($N' = N(1-\alpha)$)

3.4.2 Tree Structure Construction.

Initially, we can construct a basic balanced binary tree, as illustrated in Figure 4 (a). For optimization, we perform *adaptive node reduction*, where the reduction of a node refers to removing it from the tree and linking its child nodes to its parent. For example, reducing node n_8 in Figure 4 (a) produces the tree in Figure 4 (b). We examine all non-leaf nodes through postorder traversal, adaptively determining whether to reduce each node to minimize the average error in response to range queries.

Specifically, the average error for all possible queries can be evaluated by accumulating the noise error from nodes within query ranges and the PL fitting error from intersecting leaf nodes. Since the PL fitting error primarily depends on estimates from phase 1 (which are determined), we focus on the noise error of nodes, as formalized in the left part of Eq.(7). Here, $\mathcal{N}$ denotes all nodes in the PriPL-tree, w_k denotes the probability of node n_k being involved in queries, and α_k denotes the ratio of allocated users for node n_k. As the variance of OUE used in node frequency estimation is inversely proportional to its allocated user ratio, we simplify Err to the right part of Eq.(7). Assuming all queries arrive with equal probabilities, the weight w_k is calculated by Eq. (8) [29], where $[l_k, r_k]$ ($[l_p, r_p]$) denotes the interval of node n_k (its parent n_p), and the total number of possible queries is $(d+1)d/2$.

$$Err = \sum_{n_k \in \mathcal{N}} w_k \cdot Var(\bar{f}_k) \propto \sum_{n_k \in \mathcal{N}} w_k/\alpha_k \quad (7)$$

$$w_k = \left(l_k \cdot (d - r_k + 1) - l_p \cdot (d - r_p + 1) \right) / \left((d+1)d/2 \right) \quad (8)$$

We provide an example in Figure 4 to calculate errors and decide whether to reduce n_8. Considering that the existence of a non-leaf node only influences user allocation and weight calculation for its ancestor and descendant nodes, as framed by a red dash line in Figure 4, we compare the accumulated error from these nodes rather than the entire tree. Finally, we reduce n_8 for a smaller error.

3.5 PriPL-Tree Refinement

To address the frequency inconsistencies outlined in Section 3.2, we perform frequency and slope refinements as follows.

3.5.1 Frequency Refinement.

As we know, several methods can address these inconsistencies individually, such as Norm-Sub for issue (1) [40] and constrained inference for issue (2) [16, 29]. However, applying one method may lead to the emergence of another inconsistency. AHEAD [7] employs these two techniques iteratively to solve issues (1) and (2), but this is inefficient and often inaccurate. To address all three inconsistency issues simultaneously, we devise an optimized constrained inference method comprising two steps: *weighted averaging* and *frequency consistency*, each requiring only a single tree traversal.

In the *weighted averaging* step, we traverse nodes from leaves to root, minimizing frequency variance for each node and addressing inconsistency issue (3). For a leaf node n_k with two frequency estimates, $\hat{f}_k$ (from node frequency estimation) and $\bar{f}_k$ (from the noisy histogram estimation), we update its frequency to $\dot{f}_k = \theta \hat{f}_k + (1-\theta)\bar{f}_k$, where $\theta = Var(\bar{f}_k)/(Var(\hat{f}_k)+Var(\bar{f}_k))$, achieving the minimal variance $Var(\dot{f}_k) = Var(\hat{f}_k) Var(\bar{f}_k)/(Var(\hat{f}_k)+Var(\bar{f}_k))$. For non-leaf node n_k, we similarly update its frequency to $\dot{f}_k$, using its frequency $\hat{f}_k$ and the sum of its child nodes' frequencies $\sum_{n_c \in child(n_k)} \dot{f}_c$.

In the *frequency consistency* step, we update frequencies from the root to leaves to address inconsistency issues (1) and (2). The root's frequency is fixed at 1, i.e., $\tilde{f}_{root} = 1$. Given a parent node n_p with optimized frequency $\tilde{f}_p \geq 0$ and its child nodes' to-be-optimized frequencies $\{\dot{f}_c | n_c \in child(p)\}$, we define the optimization problem in Eq.(9). Let D_+ be the set of child nodes with positive updated frequencies, and D_0 be those with zero updated frequencies. According to KKT condidtions, the optimal frequency is $\tilde{f}_c = \dot{f}_c + \left(\tilde{f}_p - \sum_{n_c \in D_+} \dot{f}_c \right) / |D_+|$ if $n_c \in D_+$, and $\tilde{f}_c = 0$ if $n_c \in D_0$.

$$\min \sum_{n_c \in child(n_p)} (\tilde{f}_c - \dot{f}_c)^2$$
$$\text{s.t.} \quad \sum_{n_c \in child(n_p)} \tilde{f}_c = \tilde{f}_p \ \text{ and } \ \forall n_c \in child(n_p), \tilde{f}_c \geq 0 \quad (9)$$

3.5.2 Slope Refinement.

To meet frequency constraints within nodes, i.e., resolving inconsistency issue (1), we refine slopes based on optimized node frequencies. For node n_k with frequency $\tilde{f}_k$, we must guarantee non-negativity at both endpoints of interval I_k. Denoting the endpoints of I_k as l_k and r_k, we require $f(l_k) = \tilde{f}_k/|I_k| - \tilde{\beta}_k(|I_k|-1)/2 \geq 0$ and $f(r_k) = \tilde{f}_k/|I_k| + \tilde{\beta}_k(|I_k|-1)/2 \geq 0$. This establishes an effective range $[-C_k, C_k]$ for its slope, where $C_k = 2\tilde{f}_k/|I_k|(|I_k| - 1)$. We then update $\tilde{\beta}_k$ using Eq.(10), minimizing the error $(\tilde{\beta}_k - \hat{\beta}_k)^2$ and ensuring all fitted frequencies in n_k are non-negative.

$$\tilde{\beta}_k = \begin{cases} -C_k, & \hat{\beta}_k < -C_k \\ \hat{\beta}_k, & -C_k \leq \hat{\beta}_k \leq C_k \\ C_k, & \hat{\beta}_k > C_k \end{cases} \quad (10)$$

3.6 Privacy and Performance Analysis

In this subsection, we analyze the privacy guarantee, estimation error, and space and time complexity of PriPL-Tree for range queries.

3.6.1 Privacy Analysis.

During the PriPL-Tree estimation, we collect and estimate frequencies twice from users via SW and OUE, each employing non-overlapping subsets of users and utilizing the full privacy budget ϵ. The PL-fitting, PriPL-Tree construction (excluding node estimation), and refinement are post-processing steps over these collected data. As such, our PriPL-Tree satisfies ϵ-LDP.

3.6.2 Error Analysis. Range query errors from PriPL-Tree arise from two sources: *noise and sampling error* and *PL estimation error*.

Noise and sampling error arise from the frequency estimation via LDP mechanisms using a subset of users. In the private PL fitting phase (phase 1), the estimated frequency $\hat{f}_k$ of leaf node n_k, derived via the SW mechanism [25], exhibits bias depending on the data distribution and has a square error of $O(\frac{|I_k|}{N\alpha\epsilon^2})$ [8]. In the PriPL-Tree construction phase (phase 2), the node frequency $\bar{f}_k$, estimated via the OUE mechanism [38], is unbiased and has variance $\text{Var}(\bar{f}_k) = \frac{4e^\epsilon}{\alpha_k N \cdot (e^\epsilon - 1)^2} = O(\frac{1}{\alpha_k N\epsilon^2})$, where α_k represents the proportion of users allocated to node n_k. During the PriPL-Tree refinement (phase 3), we aggregate all these frequency estimates to minimize the variance of each node's frequency, leading to error bounds presented in Theorem 3.1. For the precise square error needed for multi-dimensional grid consistency refinement (Section 4.3), we introduce a numerical method. We treat the updated frequency after refinement as a weighted average of nodes' frequency estimates from the first two phases. By accounting for specific weights, we can derive accurate errors. Due to space constraints, we elaborate on this numerical method, and the proof of Theorem 3.1 in Appendix C in [36].

THEOREM 3.1. *Given a PriPL-Tree with at most K segments (corresponding to K leaf nodes), the error variance of frequencies after weight averaging in refinement (phase 3) is $O\left(\frac{K \cdot \log K}{(1-\alpha)\cdot(K+1)\cdot N \cdot \epsilon^2}\right)$ for non-leaf nodes and $O\left(\frac{\log K}{(1-\alpha)\cdot N \cdot \epsilon^2}\right)$ for leaf nodes. After frequency consistency, these variance is capped at $O\left(\frac{K \cdot \log K}{(1-\alpha)\cdot N \cdot \epsilon^2}\right)$.*

PL estimation error arises when estimating the frequency of the sub-range $[l_{\text{sub}}, r_{\text{sub}}]$ within a leaf node n_k under a linear assumption. For each PL estimated frequency $f(v)$ of value v in this subrange, its square error can be approximated by $\mathbb{E}((f(v)-f_v)^2) = \mathbb{E}((f(v) - \hat{f}_v^H) + (\hat{f}_v^H - f_v))^2) \leq 2(\mathbb{E}(f(v) - \hat{f}_v^H)^2 + \mathbb{E}(\hat{f}_v^H - f_v)^2)$. The first term represents the square error of the PL function fitting the noisy histogram, while the second term represents the noise error of the noisy histogram. The magnitude of this error depends on the noisy histogram's distortion degree, the segment number, and the actual data distribution. It tends to be small for smooth distributions and large for jagged distributions. Ultimately, the total error of $Q([l_{\text{sub}}, r_{\text{sub}}])$ accumulates the error of values in it, leading to a result proportional to the range size square $(r_{\text{sub}} - l_{\text{sub}} + 1)^2$.

3.6.3 Space and Time Complexity Analysis. Assuming PriPL-Tree's maximum segment number, i.e., the number of leaf nodes, is K, the space complexity includes the size of PriPL-Tree, $O(K)$, and the size of parameter matrices $\mathbf{X}$ and $\mathbf{A}$ for private PL fitting, $O(K \cdot d)$, which is $O(K \cdot d)$ in total. The time complexity of PriPL-Tree involves two parts — the construction time and the query time. Overall, the construction time complexity mainly arises from private user data aggregation, frequency estimation, and private PL fitting during phases 1 and 2, totaling $O(N \cdot K + d \cdot T + d \cdot \log d \cdot K^3)$. Here, T denotes the number of iterations for EM and EMS in the SW mechanism during distribution estimation in phase 1. The query time complexity, proportional to the tree height, is $O(\log_2 K)$. Due to space limitations, we provide a detailed analysis of the time complexity for each phase in Appendix E.1 of [36].

4 EXTENSION WITH ADAPTIVE GRIDS FOR MULTI-DIMENSIONAL QUERIES

Building on insights from existing works summarized in Section 2.4, we combine 1-D PriPL-Trees and 2-D grids to handle multi-dimensional range queries. In contrast to uniformly constructed 2-D grids proposed by HDG [45], we introduce data-aware adaptive grids. These grids leverage the accurate 1-D marginal distributions from PriPL-Trees to dynamically partition the entire domain into dense or sparse cells, adapting to data distribution density. As a result, they offer a more precise representation of 2-D data distributions and more accurate range query responses.

In this section, we first outline the workflow for multi-dimensional range queries and then present the core methods of *adaptive grid partitioning* and *consistency refinement*. Due to space limitations, we analyze the estimation error and runtime complexity in Appendix D and Appendix E.2 in our full version of paper [36], respectively.

4.1 Workflow of Multi-dimensional Cases

We list the workflow of multi-dimensional range queries below and provide a figure illustration in Appendix B.1 of [36].

Step 1: Estimating m 1-D Histograms using PriPL-Tree. Initially, we allocate half of the users to estimate 1-D marginal distributions. For each attribute A_i, we employ $N/2m$ users to estimate the PriPL-Tree and generate histograms $\tilde{\mathbf{F}}_i$ using the PL function within the PriPL-Tree. These histograms enable us to depict each attribute's underlying data distribution effectively.

Step 2: Estimating $\binom{m}{2}$ 2-D Adaptive Grids. For each attribute pair $\langle A_i, A_j \rangle$ ($i, j \in [m]$ and $i \neq j$), we construct a 2-D grid, as presented in Section 4.2. During each grid's construction, we partition the domain of each dimension into non-uniform intervals based on the marginal distribution, thereby forming 2-D grids that are denser in high-frequency regions and sparser in low-frequency regions, as depicted in Figure 5. After construction, we assign $N/2\binom{m}{2}$ users to estimate the frequencies of cells in each grid using the OUE mechanism with a privacy budget of ϵ.

Step 3: Refining Consistency Between Grids and PriPL-Trees. Due to the noise introduced by the LDP mechanism, frequency inconsistencies for one attribute may arise among grids and PriPL-Trees, and some frequencies could be negative. To address these issues, we propose a post-processing method, detailed in Section 4.3, optimizing the frequencies of grids and adjusting both the frequencies and slopes of nodes in PriPL-Trees.

Step 4: Answering Range Queries. For 1-D queries, we can directly utilize PriPL-Trees to respond. For a 2-D query involving attributes $\langle A_i, A_j \rangle$, we answer it by a response matrix with $d_i \times d_j$ values. This matrix represents the 2-D data distribution and is estimated from 1-D histograms and 2-D adaptive grids using the maximum entropy algorithm or weighted update as described in [37, 45]. The critical difference between us and [37, 45] is that they enforce the frequency sum of a sub-region equal to match its corresponding 1-D cell, while we enforce each frequency in the marginal distribution of the matrix to match the 1-D histogram. This fine-grained consistency fully exploits the slope information in PriPL-Trees, yielding a more accurate distribution. Further, for λ-D ($\lambda > 2$) queries, we estimate with a 2^λ response matrix based on associated 2-D queries as in [37, 45].

Marginal Distribution for A_i from PriPL-Tree

(a) Adaptive Grid on $\langle A_i, A_j \rangle$ (b) Adaptive Grid on $\langle A_i, A_{j'} \rangle$
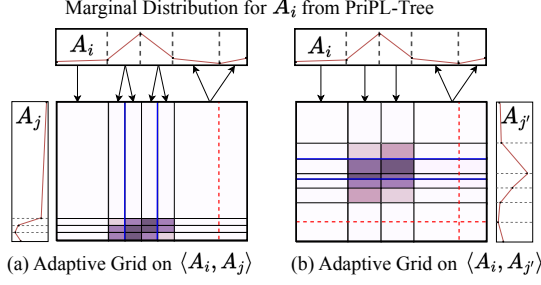
Figure 5: Examples of Adaptive Grids (Black solid lines represent partitions inherited from PriPL-Trees; blue solid lines indicate newly added partitions; red dashed lines indicate deleted partitions.)

4.2 Adaptive 2-D Grid Partitioning

The 2-D grid depicts the underlying data distribution by assuming uniform frequency distribution within each cell. To enhance its data depiction capability, we partition the grid based on data density. In data-dense areas, densely partitioned cells provide a more accurate distribution representation, reducing reliance on uniform assumptions. Conversely, in data-sparse areas, low cell frequencies may be overwhelmed by OUE noise, diminishing their effectiveness. Therefore, in such areas, sparsely partitioned cells covering larger areas with relatively higher frequencies are preferable. Building on this concept, we introduce adaptive partitioning for 2-D grids, as illustrated in Figure 5. For the attribute pair $\langle A_i, A_j \rangle$, we initially partition its domain according to the leaf node partitions in the PriPL-Trees, creating an initial grid G with $g_i \times g_j$ cells. We then dynamically adjust the partition lines, adding lines in high-frequency regions and removing them in low-frequency regions along each dimension, as indicated by the blue and red lines in Figure 5. The goal is to minimize the squared error Err_G for range queries within the grid while ensuring each cell's frequency exceeds the standard deviation of the OUE noise, $\sqrt{2\sigma^2 \cdot \binom{m}{2}}$.

We introduce the squared error Err_G as below and detail the algorithm for adaptive partitioning in Appendix B.2 in [36]. For a cell c in grid G, when fully covered by a query rectangle, it incurs a squared noise and sampling error of $2\sigma^2 \binom{m}{2}$. If the cell intersects with a query rectangle, it incurs an estimation error proportional to the intersecting frequency f_c, which can estimated by the product of its marginal frequencies. Let $\pi_i(\cdot)$ denotes the projection of the cell index on marginal attribute A_i, the squared estimation error is thus $\eta \cdot (\tilde{f}_{\pi_i(c)} \cdot \tilde{f}_{\pi_j(c)})^2$, with η as a constant. For a range query Q selecting a portion r of the area of a grid with $g_i \times g_j$ cells, the total squared error combines noise and sampling errors from rg_ig_j cells and estimation errors proportional to r times the square of all cells' frequencies, totaling $2rg_ig_j\sigma^2\binom{m}{2} + r\eta \sum_{c \in G}(\tilde{f}_{\pi_i(c)} \cdot \tilde{f}_{\pi_i(c)})^2$. Our experiments demonstrate that an η value of 0.04 provides accurate estimations across various datasets.

4.3 Consistency Refinement

For partitions of A_i between the PriPL-Tree and the grid for $\langle A_i, A_j \rangle$, we observe several one-to-many relationships, as shown by arrows in Figure 5. We treat each one-to-many relationship as a tree and can

apply our optimized constrained inference method, detailed in Section 3.5, to ensure their frequency consistency and non-negativity. In a global view, each attribute A_i is linked to $m-1$ grids, and each grid on $\langle A_i, A_j \rangle$ associates with two attributes. Applying the above method straightforwardly to update A_i and each related grid sequentially is challenging for in maintaining global consistency. For instance, resolving consistency between A_i and $\langle A_i, A_{j'} \rangle$ might reintroduce inconsistencies between A_i and $\langle A_i, A_j \rangle$ that were previously resolved. Therefore, we first update all 1-D attributes, i.e., the leaf node frequencies in PriPL-Trees, by applying improved constrained inference sequentially across their $m-1$ corresponding grids. Using these updated 1-D frequencies, we then update the grids with the frequency consistency operation (i.e., the second step in the improved constrained inference). When multiple marginal cells in grids correspond to a single leaf node, we directly use the frequency of this leaf node to update the relevant cells. Conversely, if a marginal cell in grids corresponds to multiple leaf nodes, we update the cells' frequencies using the sum of frequencies from these leaf nodes. For a grid on $\langle A_i, A_j \rangle$, to prevent reintroducing inconsistencies with A_i after aligning with A_j, we alternately update it with A_i and A_j until convergence.

Moreover, using the updated frequencies of each attribute, we can further refine the PriPL-Tree to enhance accuracy for 1-D range queries. The leaf node slope can be updated based on these frequencies as described in Section 3.5, and non-leaf node frequencies can be updated by aggregating the frequencies of their child nodes.

5 EVALUATION

In this section, we evaluate the performance of PriPL-Tree and its extension for both 1-D and multi-D range queries.

5.1 Experimental Setting

Competitors. We compare our methods with state-of-the-art techniques for range queries in LDP, including DHT [4], AHEAD [7], PrivNUD [37] for 1-D cases, and HDG [45], AHEAD, PrivNUD, PRISM [41] for multi-D cases. We exclude hierarchical tree HH [4] and HIO [39] baselines, as they have been demonstrated inferior to DHT and AHEAD in 1-D cases [4, 7] and to HDG in multi-D cases [45]. For a fair comparison, we implement these methods using the codes and parameters from their original papers.

Datasets. We use four synthetic (Gaussian, MixGaussian, Cauchy, Zipf) and four real-world datasets (Adult [1], Loan [18], Salary [19], and Financial [20]), each with 5 dimensions. On each dimension, the synthetic datasets Gaussian, Cauchy, and Zipf sample data from $Gaussian(0, 1)$, $Cauchy(0, 1)$, and $Zipf(1.1)$ distributions, respectively. The MixGaussian dataset follows a mixture of $Gaussian(0, 0.5)$ and $Gaussian(3, 0.8)$. For applying LDP mechanisms for frequency estimation and aligning with the existing works [4, 7, 37, 41, 45], all datasets are bucketized into domain [1024] for 1-D evaluations and [256] for multi-D evaluations, except for some attributes in the real-world dataset with discrete values and original domain sizes less than our specified ones. We provide statistics of these datasets in Table 2 where the mean and variance are for the default attribute in 1-D scenarios. A more detailed description is provided in Appendix F of [36].
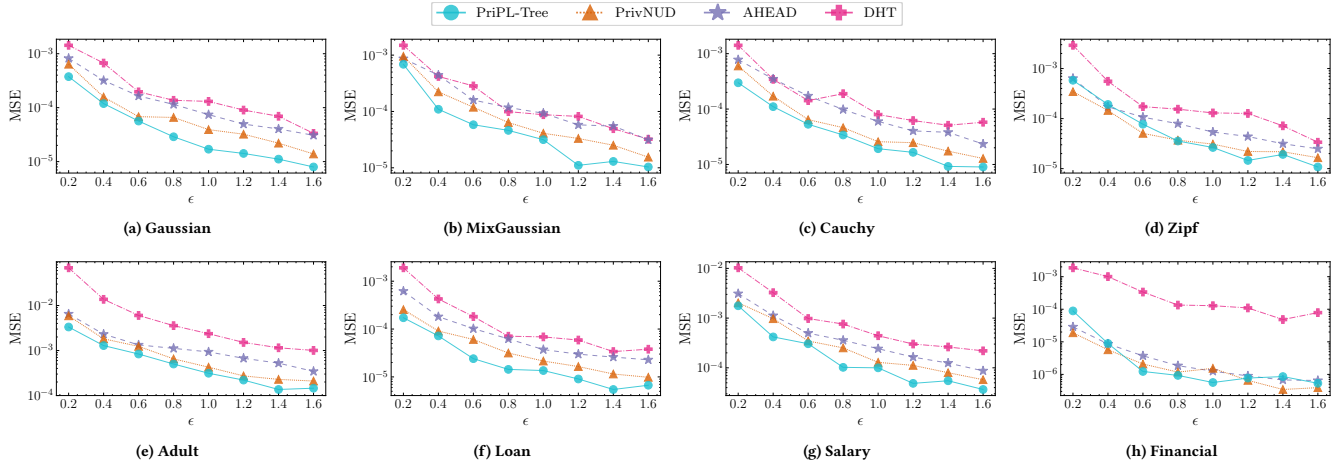
Figure 6: Evaluation for 1-D Range Queries with Varying Privacy Budget ϵ
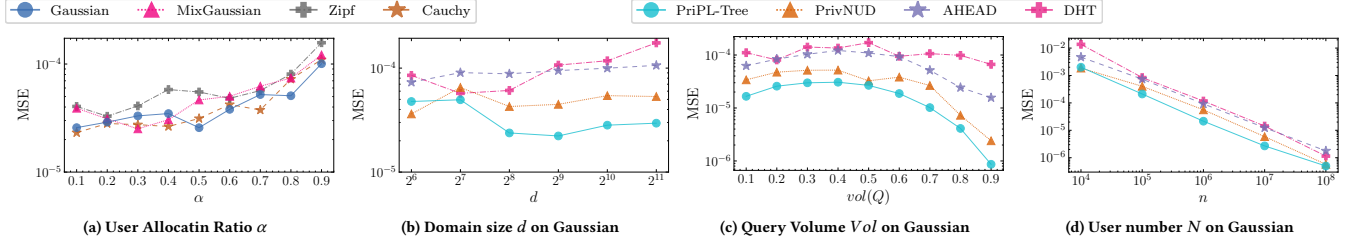

Figure 7: Evaluation for 1-D Range Queries with Varying Parameters

Metrics. We employ the mean square error (MSE) [7, 37] to quantify the deviation between the estimated ($\tilde{f}_Q$) and actual (f_Q) answers to range queries **Q**, denotes as $MSE(\mathbf{Q}) = \sum_{Q \in \mathbf{Q}}(f_Q - \tilde{f}_Q)^2/|\mathbf{Q}|$. During each evaluation, we test 1,000 randomly generated range queries with a specified query volume and report the final MSE by an average of 20 repeats of the experiment.

Default Settings. By default, we use a user allocation ratio $\alpha = 0.2$ and an acceleration granularity factor $\phi = 127$ for PriPL-Trees. For experiments, we set the privacy budget $\epsilon = 0.8$ and the query volume $vol(Q) = 0.5$, where query volume represents the ratio of the query range size to the domain size on each attribute. All experiments use Python 3.11 on a Linux server with an Intel® Xeon® Gold 5218 CPU (2.3GHz) and 96GB of memory.

5.2 1-D Experimental Results

In this subsection, we evaluate the performance of PriPL-Tree and its competitors (DHT, AHEAD, and PrivNUD) for 1-D range queries and analyze the impact of data and query parameters on them.

Overall Performance. We evaluate PriPL-Tree against three competitors across varying privacy budgets on synthetic and real-world datasets in Figure 6. Our PriPL-Tree consistently outperforms competitors on most continuous distributions, such as Gaussian, MixGaussian, Cauchy, and those in the Adult, Loan, and Salary datasets. It significantly reduces MSEs by about 12.1% to 66.6%, averaging a 37.4% reduction across different privacy settings. In highly

Table 2: Summary of Datasets

Dataset	U.#[a]	L.#[b]	Mean	Var.	Dataset	U.#	L.#	Mean	Var.
Gaussian	10^6	0	155.0	775.34	Adult	32,561	4	30.36	336.88
MixGaussian	10^6	0	135.12	2516.62	Loan	148,045	3	48.67	1620.19
Cauchy	10^6	5	159.37	609.50	Salary	2,013,799	2	33.59	515.88
Zipf	10^6	5	24.40	2517.47	Financial	6,362,620	5	0.23	2.65

[a] U.#: The number of users (i.e., samples).
[b] L.#: The number of leptokurtic attributes with a kurtosis exceeding 3.

leptokurtic distributions, like those in Zipf and Financial datasets, PriPL-Tree matches the performance of the leading competitor, PrivNUD. For these distributions, where a few values have significant frequencies, PriPL-Tree almost degenerates into an optimized hierarchical tree, similar to PrivNUD. It segregates high-frequency buckets into individual leaf nodes and merges low-frequencies into a single node, with both frequency and slope nearing zero.

Impact of User Allocation Ratio α. In Figure 7 (a), we assess the impact of the user allocation ratio α used in phase 1 of PriPL-Tree across four synthetic datasets. MSE remains stable for $\alpha \leq 0.5$ and slightly increases for $\alpha > 0.5$, suggesting that fewer users are adequate for accurate PL fitting. Thus, we empirically set $\alpha = 0.2$.

Impact of Domain Size d. In Figure 7 (b), we explore the impact of domain size d on the 1-D Gaussian dataset, demonstrating PriPL-Tree's superiority, particularly in large domains. Notably, PriPL-Tree performs less effectively in very small domains, where
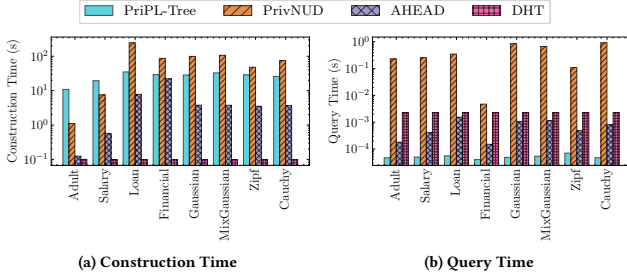
Figure 8: Runtime Evaluation under Different Datasets

coarse bucketizing reduces the histogram's accuracy in representing distributions, resulting in suboptimal PL functions and inferior outcomes.

Impact of Query Volume $vol(Q)$**.** In Figure 7 (c), we assess the impact of query volume on a Gaussian dataset. PriPL-Tree consistently records the lowest MSE. All methods display an MSE that increases initially and then decreases, peaking around $vol(Q) = 0.5$. As detailed in Section 3.6.2, the error for range queries correlates with the query range size and the frequency of intersections between query ranges and leaf nodes. Below $vol(Q) = 0.5$, MSE increases primarily due to the expanding query range size. Above $vol(Q) = 0.5$, MSE decreases as intersections occur more frequently at the domain's margins, where frequencies are lower and nearing 0, resulting in fewer PL fitting errors.

Impact of User Number N**.** In Figure 7 (d), we examine the impact of user numbers with a Gaussian dataset. MSEs decrease as user numbers increase, aligning with the law of large numbers. However, PriPL-Tree's advantage diminishes with very small (e.g., 10^4) or very large (e.g., 10^8) user numbers. Insufficient users introduce excessive noise in LDP estimation, compromising PL parameter accuracy. Conversely, a larger user pool mitigates LDP noise, enabling even simple hierarchical trees to provide accurate estimates.

Runtime Comparison: In Figure 8, we compare the runtime of our method with competitors across various datasets, with all methods implemented in Python for consistency. Our construction time is generally under half a minute. On average, our construction time of 27.2s is shorter than the average of our three competitors of 29.8s. This advantage is mainly due to the concise tree structure of PriPL-Tree, which utilizes fewer nodes. Additionally, our average query time is significantly lower, at around $50\mu s$, while our competitors' times remain in the millisecond range.

5.3 Multi-D Experimental Results

For multi-dimensional scenarios, we evaluate the performance of our method, PriPL-Tree with adaptive grids, against four competitors (HDG, AHEAD, PrivNUD, and PRISM). Typically, the standard experimental setup is evaluating 2-D queries on 5-D datasets, as all high-dimensional query results are derived from these 2-D queries.

Overall Performance. We evaluate the PriPL-Tree method against competitors under various privacy budgets on 5-D synthetic and real-world datasets, as shown in Figure 9. Utilizing adaptive grids, the PriPL-Tree method achieves the lowest MSEs in most cases, averaging 47.9% lower MSE on real-world datasets and 23.7% lower on synthetic datasets compared to state-of-the-art solutions.

The extent of improvement of PriPL-Tree varies with the characteristics of different data distributions. In the Gaussian, MixGaussian, Loan, and Salary datasets, where most attributes are not leptokurtic, our PriPL-Tree method reduces MSE by 10.6% to 81.9%, averaging a reduction of 56.7%. Conversely, for the other datasets, including Cauchy, Zipf, Adult, and Financial, which feature predominantly leptokurtic distributions where only a few values have significant frequencies, PriPL-Tree performs similarly to an optimized hierarchical tree, yielding a modest average MSE reduction of 14.9%.

Impact of Data Dimension m**.** In Figure 10 (a), we evaluate PriPL-Tree and competitors across varying data dimensions on the Gaussian dataset with a covariance of 0.6. PriPL-Tree consistently shows the lowest MSEs across different dimensions. As expected, all MSEs increase with the dimension m as users are distributed among more parts for estimation. Notably, two data points for AHEAD at $m \in \{25, 30\}$ are missing in the figure due to exceeding our server's 96GB memory capacity, as per their open-source code. Despite these omissions, the available data points are sufficient to demonstrate that AHEAD's performance is inferior to ours.

Impact of Query Dimension λ**.** In Figure 10 (b), we assess PriPL-Tree and other methods across varying query dimensions on a Gaussian dataset with $Cov = 0.6$. PriPL-Tree consistently records the lowest MSE, particularly noticeable in 1-D queries. During these experiments, we construct data structures across all five dimensions, with each 1-D range query selecting a dimension at random. This setup requires dividing users among $5 + \binom{5}{2} = 15$ parts, leading to fewer users per dimension and generally poorer 1-D estimations for competitors like HDG, AHEAD, PrivNUD, and PRISM. In contrast, our robust PriPL-Tree, enhanced by the consistency refinement in phase 3, effectively improves accuracy by updating frequencies and slopes in PriPL-Trees using all related 2-D adaptive grids.

Impact of Attribute Correlation. In Figures 10 (c) and (d), we examine the impact of attribute correlation on range queries using Gaussian and MixGaussian datasets. We use covariance to represent attribute correlation. The results reveal that PriPL-Tree consistently achieves the lowest MSEs, especially in datasets with high attribute correlations. This highlights our method's superior capability to capture underlying distributions with adaptive 2-D grids, unlike competitors that rely on uniform grids.

6 RELATED WORK

In this section, we review related works in both central differential privacy (DP) and LDP scenarios.

Range Query under DP: In DP scenarios, the far-reaching hierarchical tree and constrained inference method were first proposed by Hay et al. [16] and later optimized by Qardaji et al. [28]. Various optimizations have since been proposed to mitigate noise errors on trees: Xiao et al. [43] enhanced trees using Haar wavelet transforms; Cormode et al. [5] proposed a geometric privacy budget allocation method; Li et al. [22] optimized the non-uniform domain partitioning and privacy budget allocation based on data distribution and query workloads; Zhang et al. [49] proposed PrivTree (i.e., a Quad-tree) with optimized node decomposition; Huang et al. [17] employed a balanced box-decomposition tree (BBD-tree) for counting arbitrarily shaped geometric ranges. Beyond hierarchical trees, Qardaji et al. [28] presented the grid method with optimized
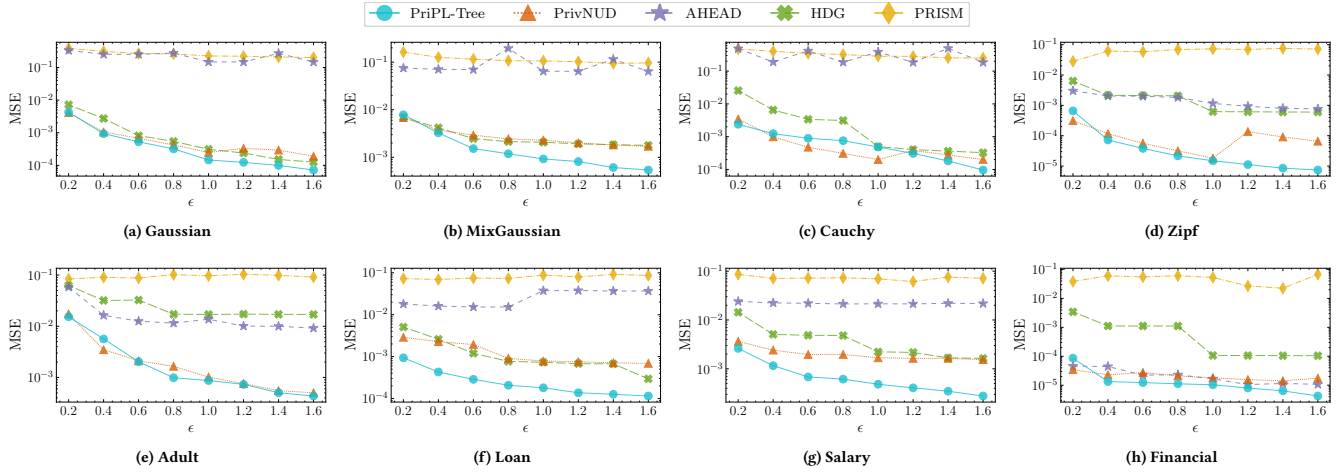
Figure 9: Evaluation for 2-D Range Queries on 5-D Datasets with Varying Privacy Budget ϵ
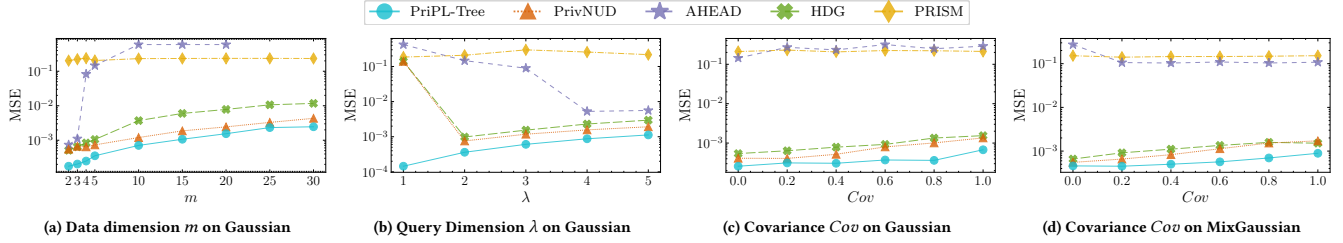

Figure 10: Evaluation for Multi-D Range Queries

granularity, claiming grids are more suitable for high-dimensional queries. Recently, Zeighami et al. [48] introduced a model-driven approach that learns noisy answers from multiple 2-D range count queries to predict results without complex indexes.

Range Query under LDP: In this context, we summarize existing methods according to tree-based and grid-based methods. In tree-based methods, HH [4] and HIO [39] proposed the basic hierarchical tree in LDP almost simultaneously and optimize the tree's fan-out (i.e., branching factor). As improvements, AHEAD [7] merged intervals with low frequencies; PrivNUD [37] customized the fan-out for each node; DHT [4] optimized this tree via Haar wavelet transformation as in [43]. In grid-based methods, HDG [45] proposed the state-of-the-art hybrid dimensional grids, and PRISM [41] replaced simple grids with prefix-sum (PS) cubes. Additionally, there are other research topics covering range questions. McKenna et al. [27] proposed a general matrix mechanism for linear queries in LDP, which can also be applied to answer range queries. And Ye et al. [47] explored PrivKVM* for range-based estimation in key-value datasets.

Marginal Release under LDP: As a relevant problem to this work, we also review marginal release methods in LDP. Cormode et al. [3] introduced a Fourier transform-based method for private marginal release. Ren et al. [32] explored multi-dimensional joint distribution estimation using the Expectation-Maximization (EM) algorithm and Lasso regression. A more advanced method is CALM, proposed by Zhang et al. [51], which extends the idea of PriView

[30] from central DP to LDP and reconstructs high-dimensional marginals using low-dimensional estimations. This idea has been widely adopted in multi-dimensional range queries.

7 CONCLUSION

In this paper, we propose the PriPL-Tree to accurately answer range queries on arbitrary data distributions. The key idea is to approximate the underlying distribution using piecewise linear functions, which alleviates both non-uniform error and LDP noise error. We further extend this with adaptive grids to handle multi-dimensional cases, where the grids dynamically adjust to the data density, thus more accurately modeling the 2-D distribution and improving accuracy for multi-dimensional range queries. Extensive experiments on both real and synthetic datasets demonstrate the effectiveness and superiority of PriPL-Tree over state-of-the-art solutions.

For future work, we will explore automatic and data-aware machine learning models to further enhance estimation in LDP scenarios.

REFERENCES

[1] Barry Becker and Ronny Kohavi. 1996. *Adult.* https://doi.org/10.24432/C5XW20 (15 Jul. 2024).

[2] Chiranjeeb Buragohain, Nisheeth Shrivastava, and Subhash Suri. 2006. Space Efficient Streaming Algorithms for the Maximum Error Histogram. In *Proceedings of the 23rd International Conference on Data Engineering.* 1026–1035.

[3] Graham Cormode, Tejas Kulkarni, and Divesh Srivastava. 2018. Marginal Release under Local Differential Privacy. In *Proceedings of the 2018 International Conference on Management of Data.* 131–146.

[4] Graham Cormode, Tejas Kulkarni, and Divesh Srivastava. 2019. Answering Range Queries under Local Differential Privacy. *Proceedings of the VLDB Endowment* 12, 10 (2019), 1126–1138.

[5] Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, Entong Shen, and Ting Yu. 2012. Differentially Private Spatial Decompositions. In *Proceedings of the 28th International Conference on Data Engineering.* IEEE, 20–31.

[6] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In *Proceedings of the Advances in Neural Information Processing Systems 30.*

[7] Linkang Du, Zhikun Zhang, Shaojie Bai, Changchang Liu, Shouling Ji, Peng Cheng, and Jiming Chen. 2021. AHEAD: Adaptive Hierarchical Decomposition for Range Query under Local Differential Privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security.* 1266–1288.

[8] Jiawei Duan, Qingqing Ye, and Haibo Hu. 2022. Utility Analysis and Enhancement of LDP Mechanisms in High-Dimensional Space. In *Proceedings of the 38th International Conference on Data Engineering.* 407–419.

[9] Jiawei Duan, Qingqing Ye, Haibo Hu, and Xinyue Sun. 2024. LDPTube: Theoretical Utility Benchmark and Enhancement for LDP Mechanisms in High-dimensional Space. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[10] John C Duchi, Michael I Jordan, and Martin J Wainwright. 2018. Minimax Optimal Procedures for Locally Private Estimation. *J. Amer. Statist. Assoc.* 113, 521 (2018), 182–201.

[11] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference.* Springer, 265–284.

[12] Hazem Elmeleegy, Ahmed K Elmagarmid, Emmanuel Cecchet, Walid G Aref, and Willy Zwaenepoel. 2009. Online Piece-wise Linear Approximation of Numerical Streams with Precision Guarantees. *Proceedings of the VLDB Endowment* 2, 1 (2009), 145–156.

[13] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* 1054–1067.

[14] Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-index: A Fully-dynamic Compressed Learned Index with Provable Worst-Case Bounds. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1162–1175.

[15] Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. 2019. Fiting-tree: A Data-aware Index Structure. In *Proceedings of the 2019 International Conference on Management of Data.* 1189–1206.

[16] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the Accuracy of Differentially Private Histograms Through Consistency. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1021–1032.

[17] Ziyue Huang and Ke Yi. 2021. Approximate Range Counting Under Differential Privacy. In *Proceedings of the 37th International Symposium on Computational Geometry (SoCG 2021).* Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[18] Kaggle. 2007. *All Lending Club loan data.* https://www.kaggle.com/datasets/wordsforthewise/lending-club (15 Jul. 2024).

[19] Kaggle. 2017. *SF Salaries.* https://www.kaggle.com/datasets/kaggle/sf-salaries (15 Jul. 2024).

[20] Kaggle. 2017. *Synthetic Financial Datasets For Fraud Detection.* https://www.kaggle.com/datasets/ealaxi/paysim1 (15 Jul. 2024).

[21] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. 2001. An Online Algorithm for Segmenting Time Series. In *Proceedings of the 2001 IEEE International Conference on Data Mining.* IEEE, 289–296.

[22] Chao Li, Michael Hay, Gerome Miklau, and Yue Wang. 2014. A Data- and Workload-Aware Algorithm for Range Queries under Differential Privacy. *Proceedings of the VLDB Endowment* 7, 5 (2014), 341–352.

[23] Pengfei Li, Yu Hua, Jingnan Jia, and Pengfei Zuo. 2021. FINEdex: A Fine-grained Learned Index Scheme for Scalable and Concurrent Memory Systems. *Proceedings of the VLDB Endowment* 15, 2 (2021), 321–334.

[24] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. 2020. LISA: A Learned Index Structure for Spatial Data. In *Proceedings of the 2020 International Conference on Management of Data.* 2119–2133.

[25] Zitao Li, Tianhao Wang, Milan Lopuhaä-Zwakenberg, Ninghui Li, and Boris Škoric. 2020. Estimating Numerical Distributions under Local Differential Privacy. In *Proceedings of the 2020 International Conference on Management of Data.* 621–635.

[26] Xiaoyan Liu, Zhenjiang Lin, and Huaiqing Wang. 2008. Novel Online Methods for Time Series Segmentation. *IEEE Transactions on Knowledge and Data Engineering* 20, 12 (2008), 1616–1626.

[27] Ryan McKenna, Raj Kumar Maity, Arya Mazumdar, and Gerome Miklau. 2020. A Workload-adaptive Mechanism for Linear Queries under Local Differential Privacy. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1905–1918.

[28] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Differentially Private Grids for Geospatial Data. In *Proceedings of the 29th International Conference on Data Engineering.* IEEE, 757–768.

[29] 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.

[30] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2014. Priview: Practical Differentially Private Release of Marginal Contingency Tables. In *Proceedings of the 2014 International Conference on Management of Data.* 1435–1446.

[31] Qiuyu Qian, Qingqing Ye, Haibo Hu, Kai Huang, Tom Tak-Lam Chan, and Jin Li. 2023. Collaborative Sampling for Partial Multi-Dimensional Value Collection Under Local Differential Privacy. *IEEE Transactions on Information Forensics and Security* 18 (2023), 3948–3961.

[32] Xuebin Ren, Chia-Mu Yu, Weiren Yu, Shusen Yang, Xinyu Yang, Julie A McCann, and S Yu Philip. 2018. LoPub: High-Dimensional Crowdsourced Data Publication with Local Differential Privacy. *IEEE Transactions on Information Forensics and Security* 13, 9 (2018), 2151–2166.

[33] Cosma Shalizi. 2015. *Lecture 1: Optimal Prediction (with Refreshers).* https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/01/lecture-01.pdf (15 Jul. 2024).

[34] Apple Differential Privacy Team. 2017. *Learning with Privacy at Scale.* https://machinelearning.apple.com/research/learning-with-privacy-at-scale (15 Jul. 2024).

[35] Leixia Wang, Qingqing Ye, Haibo Hu, and Xiaofeng Meng. 2023. EPS^2: Privacy Preserving Set-Valued Data Analysis in the Shuffle Model. *IEEE Transactions on Knowledge and Data Engineering* (2023), 1–14.

[36] Leixia Wang, Qingqing Ye, Haibo Hu, and Xiaofeng Meng. 2024. PriPL-Tree: Accurate Range Query for Arbitrary Distribution under Local Differential Privacy. (2024). https://github.com/LeixiaWang/PriPLT (15 Jul. 2024).

[37] Ning Wang, Yaohua Wang, Zhigang Wang, Jie Nie, Zhiqiang Wei, Peng Tang, Yu Gu, and Ge Yu. 2023. PrivNUD: Effective Range Query Processing under Local Differential Privacy. In *Proceedings of the 39th International Conference on Data Engineering.* IEEE, 2660–2672.

[38] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. 2017. Locally Differentially Private Protocols for Frequency Estimation. In *Proceedings of the 26th USENIX Conference on Security Symposium.* 729–745.

[39] Tianhao Wang, Bolin Ding, Jingren Zhou, Cheng Hong, Zhicong Huang, Ninghui Li, and Somesh Jha. 2019. Answering Multi-Dimensional Analytical Queries under Local Differential Privacy. In *Proceedings of the 2019 International Conference on Management of Data.* 159–176.

[40] Tianhao Wang, Milan Lopuhaa-Zwakenberg, Zitao Li, Boris Skoric, and Ninghui Li. 2020. Locally Differentially Private Frequency Estimation with Consistency. In *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium.* 1–16.

[41] Yufei Wang and Xiang Cheng. 2022. PRISM: Prefix-Sum Based Range Queries Processing Method under Local Differential Privacy. In *Proceedings of the 38th International Conference on Data Engineering.* IEEE, 433–445.

[42] Zhiliang Wei, Liangjie Lin, Youhe Chen, Yanqin Lin, and Zhong Chen. 2014. Partial Homogeneity Based High-Resolution Nuclear Magnetic Resonance Spectra under Inhomogeneous Magnetic Fields. *Applied Physics Letters* 105, 13 (2014).

[43] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. 2010. Differential Privacy via Wavelet Transforms. *IEEE Transactions on Knowledge and Data Engineering* 23, 8 (2010), 1200–1214.

[44] Qing Xie, Chaoyi Pang, Xiaofang Zhou, Xiangliang Zhang, and Ke Deng. 2014. Maximum Error-bounded Piecewise Linear Representation for Online Stream Approximation. *The VLDB Journal* 23 (2014), 915–937.

[45] Jianyu Yang, Tianhao Wang, Ninghui Li, Xiang Cheng, and Sen Su. 2020. Answering Multi-Dimensional Range Queries under Local Differential Privacy. *Proceedings of the VLDB Endowment* 14, 3 (2020), 378–390.

[46] Qingqing Ye, Haibo Hu, Kai Huang, Man Ho Au, and Qiao Xue. 2023. Stateful Switch: Optimized Time Series Release with Local Differential Privacy. In *Proceedings of the IEEE INFOCOM 2023 - IEEE Conference on Computer Communications.* IEEE, 1–10.

[47] Qingqing Ye, Haibo Hu, Xiaofeng Meng, Huadi Zheng, Kai Huang, Chengfang Fang, and Jie Shi. 2021. PrivKVM*: Revisiting Key-Value Statistics Estimation with Local Differential Privacy. *IEEE Transactions on Dependable and Secure Computing* 20, 1 (2021), 17–35.

[48] Sepanta Zeighami, Ritesh Ahuja, Gabriel Ghinita, and Cyrus Shahabi. 2022. A Neural Database for Differentially Private Spatial Range Queries. *Proceedings of the VLDB Endowment* 15, 5 (2022), 1066–1078.

[49] 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.

[50] Yuemin Zhang, Qingqing Ye, Rui Chen, Haibo Hu, and Qilong Han. 2023. Trajectory Data Collection with Local Differential Privacy. *Proceedings of the VLDB Endowment* 16, 10 (2023), 2591–2604.

[51] Zhikun Zhang, Tianhao Wang, Ninghui Li, Shibo He, and Jiming Chen. 2018. CALM: Consistent Adaptive Local Marginal for Marginal Release under Local Differential Privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* 212–229.

A TECHNICAL DETAILS FOR 1-D PRIPL-TREE

In this section, we detail the computation of Eq.(1) for responding to 1-D range queries and provide examples to illustrate interval partitioning and its two included strategies.

A.1 Response to 1-D Range Query

In this subsection, we focus on computing $Q([l_{sub}, r_{sub}])$ within node n_k. Given the interval $I_k = [s_{k-1}, s_k]$, slope $\tilde{\beta}_k$, and node frequency $\tilde{f}_k$ of node n_k, we first calculate the corresponding linear function $f = \tilde{\beta}_k \cdot v + b_k$, where b_k is the intercept. This function is represented by the green line in Figure 11. Because $\tilde{f}_k = \sum_{v \in I_k} (\tilde{\beta}_k \cdot v + b_k)$, we deduce that $b_k = \frac{\tilde{f}_k}{|I_k|} - \frac{\tilde{\beta}_k \cdot (2s_{k-1} + |I_k| - 1)}{2}$. Next, we compute $Q([l_{sub}, r_{sub}])$, which represents the frequency of the red-hatched region in Figure 11:

$$
\begin{aligned}
Q&([l_{sub}, r_{sub}]) \\
&= \sum_{v \in [l_{sub}, r_{sub}]} (\tilde{\beta}_k \cdot v + b_k) \\
&= \tilde{\beta}_k \cdot \frac{(l_{sub} + r_{sub}) \cdot (r_{sub} - l_{sub} + 1)}{2} + b_k \cdot (r_{sub} - l_{sub} + 1) \\
&= (r_{sub} - l_{sub} + 1) \cdot \left(\tilde{\beta}_k \left(\frac{l_{sub} + r_{sub} + 1 - |I_k|}{2} - s_{k-1} \right) + \frac{\tilde{f}_k}{|I_k|} \right).
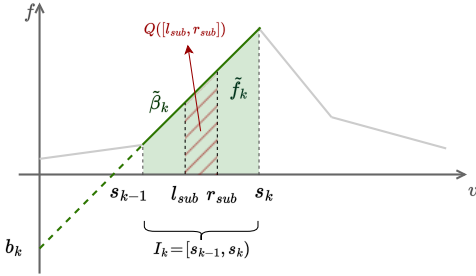\end{aligned}
$$

Figure 11: An Example of $Q([l_{sub}, r_{sub}])$ within node n_k.

A.2 Interval Partitioning

In this subsection, we illustrate the basic workflow of interval partitioning in Figure 12 (a). During interval partitioning, we have presented two strategies — twice partitioning and search acceleration — to enhance effectiveness and efficiency, respectively, in Section 3.6.2. Here, we provide additional examples of these two strategies for clarity and demonstrate their benefits.

A.2.1 Twice Partitioning Strategy. To illustrate the twice partitioning strategy, we present its results on the Cauchy and Salary datasets in Figure 13. The blue solid lines, representing $\hat{\mathbf{F}}^{EM}$ derived by EM in SW, are jagged and noisy, while the red solid lines, representing $\hat{\mathbf{F}}^{EMS}$ derived by EMS in SW, are smoother but tend to flatten peak features. Initial partitions based on $\hat{\mathbf{F}}^{EM}$ are marked by dashed blue lines and help identify critical peaks indicated by yellow stars. Subsequent partitions on $\hat{\mathbf{F}}^{EMS}$ are shown with red dashed lines, identifying critical turning points marked by black
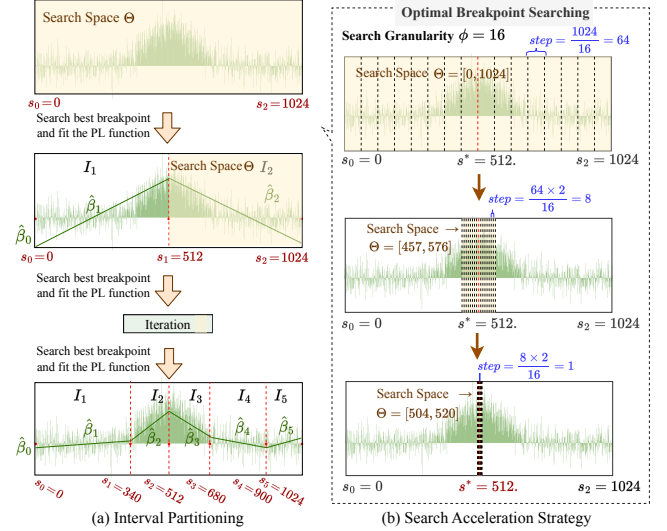

Figure 12: An Example of Interval Partitioning.

stars. In combination, this strategy facilitates the discovery of the most crucial partitions for precise PL fitting.

It is important to note that while $\hat{\mathbf{F}}^{EMS}$ does not directly provide accurate slopes at this phase, they can be refined during the PriPL-Tree refinement phase, deriving piecewise linear lines denoted by the black solid lines in Figure 13. Additionally, this strategy does not significantly increase the time complexity of interval partitioning. Partitioning over the jagged $\hat{\mathbf{F}}^{EM}$ typically converges quickly, and the partitioning over the smoother $\hat{\mathbf{F}}^{EMS}$, building on initial partitioning results, also converge swiftly. The total number of partitions remains within the maximum segment limit $K_{\max}$.
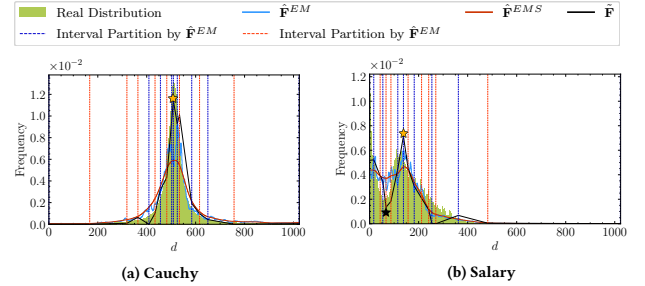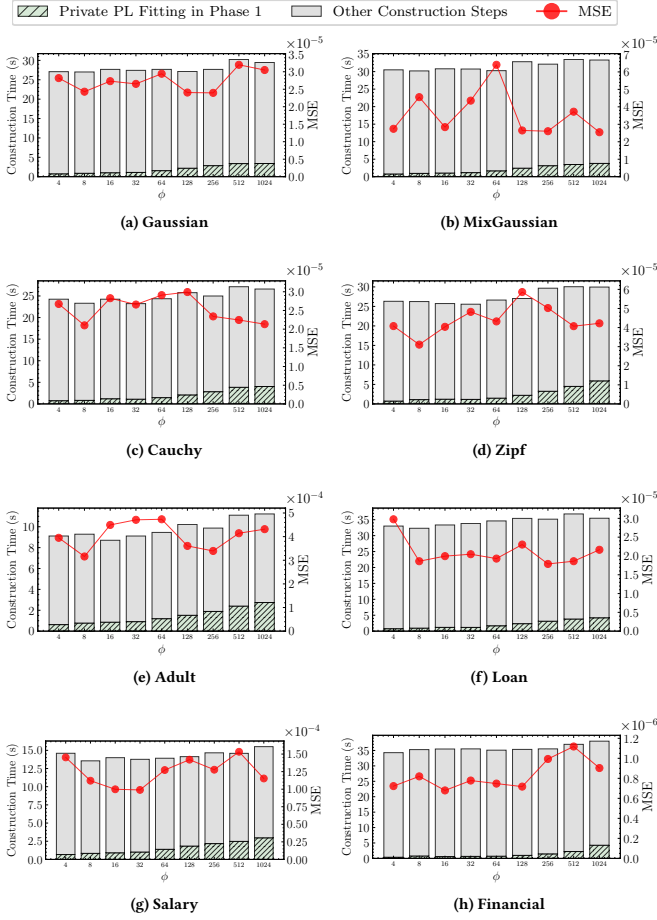

Figure 13: An Example of Twice Partitioning with $\epsilon = 0.2$

A.2.2 Search Acceleration Strategy. In Figure 12 (b), we illustrate the search acceleration strategy, which speeds up the initial breakpoint search during interval partitioning. Using a granularity factor of $\phi = 16$, we reduce the search step from 64 to 1, limiting the breakpoint candidates to no more than 16 per search. Black dashed lines mark the candidates, with the red dashed line indicating the optimal breakpoint. This method, compared to the basic one traversing all domain values, reduces the times of PL fitting for breakpoints from $O(d)$ to $O(\phi \cdot \log_\phi d)$. Although this acceleration might lead

Figure 14: Construction Time and MSE for PriPL-Tree with Search Acceleration Strategy across Different Granularity Factors ϕ

Algorithm 3: Adaptive 2-D Grid Partitioning

Input: Attribute pair $\langle A_i, A_j \rangle$ and their marginal histograms
Output: Grid G with adaptive partitions

1 Initialize a Grid G with size $g_i \times g_j$;
2 Set $S = \{I_1^i, I_2^i, \ldots, I_{g_i}^i\} \cup \{I_1^j, I_2^j, \ldots, I_{g_j}^j\}$ and $M = \phi$;
3 **while** $S \neq \phi$ **do**
4   Select marginal cell c_I ($I \in S$) with maximal frequency $\tilde{f}_I$;
5   Consider splitting c_I into c_{I_1} and c_{I_2} with $\tilde{f}_{c_{I_1}} \approx \tilde{f}_{c_{I_2}}$;
6   **if** $\tilde{f}_I > \bar{\sigma}$ *and* $Err_G(S) > Err_G(S - \{I\} \cup \{I_1, I_2\})$ **then**
7   Split c_I and update $S \leftarrow S - \{I\} \cup \{I_1, I_2\}$;
8   **else**
9   Delete I from S and add it to M;

10 **while** $M \neq \phi$ **do**
11   Select marginal cell c_I ($I \in M$) with minimal frequency;
12   **if** *the index I' of its neighbor exists in M* **then**
13   Consider merging c_I and $c_{I'}$ into c_{I_3};
14   **if** $\tilde{f}_{I_3} \leq \bar{\sigma}$ *or* $Err_G(M) > Err_G(M - \{I, I'\} \cup \{I_3\})$ **then**
15   Merging c_I and $c_{I'}$, update $M \leftarrow M - \{I, I'\} \cup \{I_3\}$;
16   Delete I' from M;
17   Delete I from M;

B TECHNICAL DETAILS FOR MULTI-DIMENSIONAL RANGE QUERIES

In this section, we illustrate the entire workflow and detail the algorithm for adaptive 2-D grid partitioning.

B.1 The Workflow

We provide an example of the workflow in Figure 15 to illustrate this process further.

B.2 The Algorithm For Adaptive Grid Partitioning

Building on the concept of adaptive partitioning described in Section 4.2, we detail the algorithm in Algorithm 3 and provide an example in step 2 of Figure 5. For the attribute pair $\langle A_i, A_j \rangle$, we initially partition its domain based on the leaf node partitions in the PriPL-Trees, creating a grid G with $g_i \times g_j$ cells (as shown in line 1). We then dynamically adjust partition lines along each dimension for adaptive partitioning, where adding a line splits a marginal cell (as shown in line 5) and removing a line merges marginal cells (as shown in line 13). In the algorithm, let I index marginal cells on both A_i and A_j; we consistently select the most significant cells for splitting (high-frequency cells) or merging (low-frequency cells). This adjustment of marginal cells adheres to two principles: ensuring each cell's frequency exceeds the standard deviation of the OUE noise $\bar{\sigma} = \sqrt{2\sigma^2 \cdot \binom{m}{2}}$, and minimizing the squared error Err_G for range queries, as dictated by the splitting and merging conditions in lines 6 and 14.

---

to a local rather than global optimal breakpoint, it has minimal impact on final results since each breakpoint in a piecewise linear function naturally represents a local optima to partition the domain. Moreover, because PL fitting occurs on a noisy histogram, a global optimal breakpoint might not accurately represent the actual distribution. Using a multi-granular search strategy that increases randomness can help mitigate this issue.

To validate the search acceleration strategy, we evaluated both the accelerated PriPL-Tree construction time and the MSE of queries on corresponding trees across four synthetic and four real-world datasets in Figure 14. A granularity factor ϕ of 1024, meaning all values in the domain are traversed, represents no acceleration. The green bar indicates the accelerated private PL fitting time, which increases with ϕ. The red line shows the MSE, fluctuating irregularly with increases in ϕ, yet the variation between the highest and lowest MSEs remained within a two-fold difference. Observationally, a larger ϕ in the range of $[128, 256]$ often provides better utility while effectively speeding up the process.
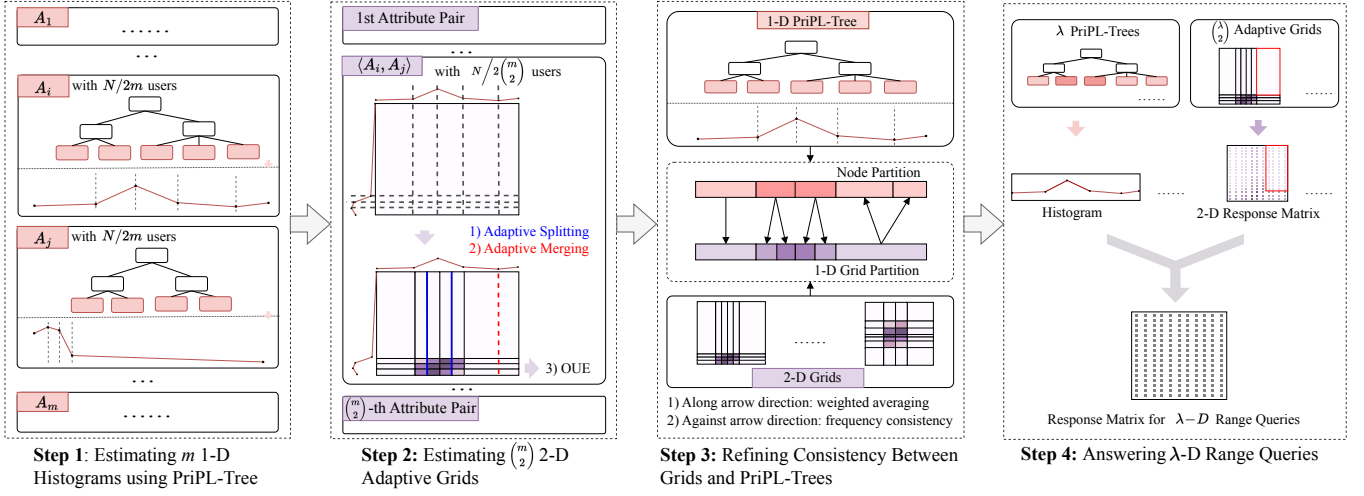
Figure 15: Workflow of Multi-dimensional Range Queries

C ERROR ANALYSIS ON 1-D PRIPL-TREE

In this section, we prove the asymptotic bound for noise and sampling error in PriPL-Tree and present a numerical method to accurately compute this error.

C.1 Proof and Analysis of Theorem 3.1

For convience, we restate the theorem as follows.

THEOREM 3.1. *Given a PriPL-Tree with at most K segments (corresponding to K leaf nodes), the error variance of frequencies after weight averaging in refinement (phase 3) is $O\left(\frac{K \cdot \log K}{(1-\alpha) \cdot (K+1) \cdot N \cdot \epsilon^2}\right)$ for non-leaf nodes and $O\left(\frac{\log K}{(1-\alpha) \cdot N \cdot \epsilon^2}\right)$ for leaf nodes. After frequency consistency, these variance is capped at $O\left(\frac{K \log K}{(1-\alpha) N \epsilon^2}\right)$.*

C.1.1 Analysis. It is important to note that after the frequency consistency step, some nodes may experience an increase in variance from $O\left(\frac{\log K}{(1-\alpha)N\epsilon^2}\right)$ to $O\left(\frac{K \log K}{(1-\alpha)N\epsilon^2}\right)$. However, most nodes exhibit reduced variances, which can be verified by computing the precise variances numerically. This phenomenon is common to any tree structure with non-uniform branchings, such as PrivNUD [37] and AHEAD [7], which exhibit heteroscedastic frequencies across nodes. In particular, in cases where the tree features uniform branching at each layer and all leaf nodes are at the same depth, each layer's nodes will be allocated an equal number of users, resulting in uniform variance among their estimated frequencies. According to the Gauss-Markov theorem [16], the variance post-frequency consistency in such scenarios can be estimated as $O\left(\frac{\log K}{(1-\alpha)N\epsilon^2}\right)$.

C.1.2 Proof. We prove the Theorem 3.1 as follows.

PROOF. For convenience, we assume the PriPL-Tree has a maximum height of $h_{\max} \leq \log_2 K$. Each node utilizes α_k users, with $\alpha_k \geq \frac{1-\alpha}{h_{\max}} \geq \frac{1-\alpha}{\log_2 K}$, and each non-leaf node has b_k branches with $1 \leq b_k \leq K$.

First, we prove the error bound after the weight averaging step.

For leaf nodes n_k, there are two frequencies: $\hat{f}_k$ derived from the SW mechanism using αN users and $\bar{f}_k$ from the OUE mechanism using α_k users. By weighted averaging, the variance of the updated frequency $\dot{f}_k$ is given by

$$\begin{aligned}
\text{Var}(\dot{f}_k) &= \frac{\text{Var}(\hat{f}_k) \cdot \text{Var}(\bar{f}_k)}{\text{Var}(\hat{f}_k) + \text{Var}(\bar{f}_k)} \\
&= \text{Var}(\bar{f}_k) - \frac{\text{Var}^2(\bar{f}_k)}{\text{Var}(\hat{f}_k) + \text{Var}(\bar{f}_k)} \\
&\leq \text{Var}(\bar{f}_k) \\
&= \frac{4e^\epsilon}{N \cdot \alpha_k \cdot (e^\epsilon - 1)^2} \\
&= O\left(\frac{1}{N\alpha_k\epsilon^2}\right) \\
&= O\left(\frac{\log K}{(1-\alpha)N\epsilon^2}\right).
\end{aligned}$$

For the non-leaf node n_p with b_p child nodes, its frequency is updated based on its own and its children's frequencies $\{\dot{f}_c | n_c \in \text{child}(n_p)\}$. Before computing the updated variance, we present two preliminary conclusions:

(1) For a child node n_c, its updated frequency variance $\text{Var}(\dot{f}_c)$ is less than or equal to the variance of the parent node's original frequency, i.e., $\text{Var}(\bar{f}_p)$. Assuming N' users are allocated to the subtree rooted at n_p and h_p denotes the subtree's maximum height. Since node n_p estimates its frequency $\bar{f}_p$ with at most N'/h_p users and its children are allocated at least N'/h_p users, the child node's frequency variance $\text{Var}(\bar{f}_c)$ does not exceed that of n_p, i.e., $\text{Var}(\bar{f}_c) \leq \text{Var}(\bar{f}_p)$. Due to weighted averaging, the updated variance $\text{Var}(\dot{f}_c) \leq \text{Var}(\bar{f}_c)$, implying $\text{Var}(\dot{f}_c) \leq \text{Var}(\bar{f}_p)$.

(2) Estimates for different child nodes $\{\dot{f}_c | n_c \in \text{child}(n_p)\}$ of node n_p are independent, as each aggregates frequencies from its respective subtree, with no overlap among these subtrees.

As such, the variance of n_p for non-leaf nodes can be deduced as follows:

$$\text{Var}(\dot{f}_p) = \frac{\text{Var}(\bar{f}_p) \cdot \text{Var}\left(\sum_{n_c \in \text{child}(n_p)} \dot{f}_c\right)}{\text{Var}(\bar{f}_p) + \text{Var}\left(\sum_{n_c \in \text{child}(n_p)} \dot{f}_c\right)}$$

$$= \frac{\text{Var}(\bar{f}_p) \cdot \left(\sum_{n_c \in \text{child}(n_p)} \text{Var}(\dot{f}_c)\right)}{\text{Var}(\bar{f}_p) + \sum_{n_c \in \text{child}(n_p)} \text{Var}(\dot{f}_c)}$$

$$= \text{Var}(\bar{f}_p) - \frac{\text{Var}^2(\bar{f}_p)}{\text{Var}(\bar{f}_p) + \sum_{n_c \in \text{child}(n_p)} \text{Var}(\dot{f}_c)}$$

$$\leq \text{Var}(\bar{f}_p) - \frac{\text{Var}^2(\bar{f}_p)}{\text{Var}(\bar{f}_p) + b \cdot \text{Var}(\bar{f}_p)}$$

$$= \frac{b_k}{b_k + 1} \cdot \text{Var}(\bar{f}_p)$$

$$\leq \frac{K}{K+1} \cdot \text{Var}(\bar{f}_p)$$

$$= O\left(\frac{K}{K+1} \cdot \frac{\log K}{(1-\alpha)N\epsilon^2}\right).$$

Next, we analyze the error variance following the frequency consistency step. Let n_c represent the child node pending update, and n_p its parent. At this stage, the nodes $n_c \in \text{child}(n_p)$ are divided into two groups, D_0 and D_+. Nodes in D_0 typically display frequencies $\dot{f}_c$ close to or below zero, which are then adjusted to zero. This modification might slightly increase or decrease their variances but does not alter their variance bound. For nodes in D_+, frequency updates are based on both parental and sibling frequencies, leading to a variance as follows.

$$\text{Var}(\tilde{f}_c)$$

$$= \text{Var}\left(\frac{|D_+| - 1}{|D_+|} \cdot \dot{f}_c + \frac{1}{|D_+|} \cdot \tilde{f}_p - \frac{1}{|D_+|} \sum_{n_s \in \text{child}(n_p)/n_c} \dot{f}_s\right)$$

$$= \frac{(|D_+| - 1)^2}{|D_+|^2} \cdot \text{Var}(\dot{f}_c) + \frac{1}{|D_+|^2} \cdot \text{Var}(\tilde{f}_p)$$

$$+ \frac{1}{|D_+|^2} \cdot \sum_{n_s \in \text{child}(n_p) \backslash n_c} \text{Var}(\dot{f}_s) + \frac{2(|D_+| - 1)}{|D_+|^2} \cdot \text{Cov}(\tilde{f}_p, \dot{f}_c)$$

$$- \frac{2}{|D_+|^2} \cdot \text{Cov}\left(\tilde{f}_p, \sum_{n_s \in \text{child}(n_p) \backslash n_c} \dot{f}_s\right)$$

$$\leq \frac{(|D_+| - 1)^2}{|D_+|^2} \cdot \text{Var}(\dot{f}_c) + \frac{1}{|D_+|^2} \cdot \text{Var}(\tilde{f}_p)$$

$$+ \frac{1}{|D_+|^2} \cdot \sum_{n_s \in \text{child}(n_p) \backslash n_c} \text{Var}(\dot{f}_s)$$

$$+ \frac{2(|D_+| - 1)}{|D_+|^2} \cdot \sqrt{\text{Var}(\tilde{f}_p) \cdot \text{Var}(\dot{f}_c)}$$

$$\leq \frac{|D_+|^2 + |D_+| - 1}{|D_+|^2} \max\left(\{\text{Var}(\tilde{f}_p)\} \cup \{\text{Var}(\dot{f}_j)|j \in \text{child}(n_p)\}\right)$$

$$\leq \frac{|D_+| + 1}{|D_+|} \max\left(\{\text{Var}(\tilde{f}_p)\} \cup \{\text{Var}(\dot{f}_j)|j \in \text{child}(n_p)\}\right)$$

$$\leq \left(\frac{|D_+| + 1}{|D_+|}\right)^{h_{\max}} \max\left(\{\text{Var}(\bar{f}_p)\} \cup \{\text{Var}(\bar{f}_j)|j \in \text{child}(n_p)\}\right)$$

$$\leq 2^{\log_2 K} \cdot O\left(\frac{\log K}{(1-\alpha)N\epsilon^2}\right)$$

$$= O\left(\frac{K \log K}{(1-\alpha)N\epsilon^2}\right).$$

$\square$

C.2 The Numerical Method for Calculating Noise and Sampling Error

The variances of node n_k's frequency $\tilde{f}_k$ after PriPL-Tree Refinement can be veiwed as a weighted average of frequency estimates of all nodes from the first two phases [29], as we exemplified in Figure 16. Here, the vector $\mathbf{V}$ stores the original independent variances of each node, and each node n_k maintains a weight vector $\mathbf{W}_k$ corresponding to each value in $\mathbf{V}$. Let $w_{k,j}$ denote the j-th value in $\mathbf{W}_k$ and v_j denote the j-th value in $\mathbf{V}$. For node n_k, its final variance can be computed as $\sum_{j \leq |\mathbf{V}|} w_{k,j}^2 \cdot v_j$.
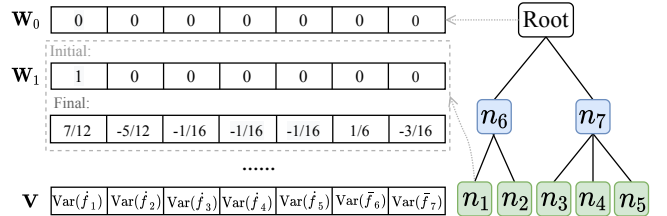
Figure 16: An Example of Variance Computation.
(For convenience, assuming all values in $\mathbf{V}$ are equal to v^* and all frequencies remain positive during the frequency consistency step, we can derive weight $\mathbf{W}_1$ as illustrated in the figure. Consequently, the final updated variance for $\tilde{f}_1$ is $\text{Var}(\tilde{f}_1) = \frac{113}{192}v^*$.)

Following this idea, the main challenges in calculating the variance of each node's frequency involve computing the basic variance vector $\mathbf{V}$ and each node's weight vector $\mathbf{W}_k$. We provide an algorithm in Algorithm 4 and present it as follows.

For the vector $\mathbf{V}$, each non-leaf node n_k records the variance $\text{Var}(\bar{f}_k)$ (i.e., OUE's variance) within it, as shown in lines 6~7 in Algorithm 4. For each leaf node n_k, it stores the variance of the weighted updated frequency $\dot{f}_k$, which is computed by combining $\text{Var}(\bar{f}_k)$ derived according to OUE and $\text{Var}(\hat{f}_k)$ derived according to SW, as shown in line 5 in Algorithm 4. Unlike OUE, which yields an unbiased estimate with constant variance, SW leads to estimates that are biased and affected by the unknown original data distribution, making it difficult to express the variance directly [25]. As an alternative, we can compute its empirical error $(\hat{f}_k - \bar{f}_k)^2$ if we approximate f_k by $\bar{f}_k$. Considering $\mathbb{E}((\hat{f}_k - \bar{f}_k)^2) = \mathbb{E}(((\hat{f}_k - f_k) + (\bar{f}_k - f_k))^2) = \mathbb{E}(\hat{f}_k - f_k)^2 + \text{Var}(\bar{f}_k)$ and that SW typically derives a better estimate with smaller variance than OUE [25], we can use $(\hat{f}_k - \bar{f}_k)^2 - \text{Var}(\bar{f}_k)$ to approximate its square error further when $(\hat{f}_k - \bar{f}_k)^2 > \text{Var}(\bar{f}_k)$. For convenience, we use this approximate square error to approximate the variance, as shown in lines 3~4 in Algorithm 4.

For the weight vector $\mathbf{W}_k$ of node n_k, after *weighted averaging*, it is updated as shown in lines 10~11 in Algorithm 4, with the optimized variance presented in Section 3.5. During *frequency consistency*, node $n_k \in D_0$, which has a frequency close to or below zero, is set to zero, and its variance changes minimally. For node $n_k \in D_+$, its updated frequency is the weighted average of itself, its parent's frequency, and its sibling $n_s \in D_+$'s frequencies. As such, its weight can be updated as indicated in lines 14~15 in Algorithm 4.

Algorithm 4: Numerical Method for Variance Estimation

Input: Frequencies $\hat{f}_k$ or $\bar{f}_k$ for each node and the node number $|\mathcal{T}|$

Output: The error variance of each node's refined frequency.

// Initialize $\mathbf{V}_{|\mathcal{T}|\times 1}$

1 Initialize vector $\mathbf{V}$;

2 **for** *leaf node n_k* **do**

3     $\mathrm{Var}(\bar{f}_k) = \frac{4e^\epsilon}{\alpha_k \cdot N \cdot (e^\epsilon - 1)^2}$;

4     $\mathrm{Var}(\hat{f}_k) \approx (\hat{f}_k - \bar{f}_k)^2 > \mathrm{Var}(\bar{f}_k)?(\hat{f}_k - \bar{f}_k)^2 - \mathrm{Var}(\bar{f}_k):$ $(\hat{f}_k - \bar{f}_k)^2$;

5     $v_k = \mathrm{Var}(\dot{f}_k) = \frac{\mathrm{Var}(\hat{f}_k)\,\mathrm{Var}(\bar{f}_k)}{\mathrm{Var}(\hat{f}_k)+\mathrm{Var}(\bar{f}_k)}$;

6 **for** *non-leaf node n_k* **do**

7     $v_k = \mathrm{Var}(\bar{f}_k) = \frac{4e^\epsilon}{\alpha_k \cdot N \cdot (e^\epsilon - 1)^2}$;

// Calculate $\mathbf{W}$

8 Initialize vector $\mathbf{W}_0 = \mathbf{0}_{|\mathcal{T}|\times 1}$ for the root and $\mathbf{W}_k$ with only the k-th value being one and others being zero for node n_k;

9 **for** *non-leaf node n_k visited in postorder traversal* **do**

10     $\theta = \frac{\sum_{n_c \in \mathrm{child}(n_k)} \mathrm{Var}(\dot{f}_c)}{\mathrm{Var}(\bar{f}_k)+\sum_{n_c \in \mathrm{child}(n_k)} \mathrm{Var}(\dot{f}_c)}$;

11     $\mathbf{W}_k = \theta \cdot \mathbf{W}_k + (1-\theta) \cdot (\sum_{n_c \in \mathrm{child}(n_k)} \mathbf{W}_c)$;

12 **for** *node n_k visited in level order traversal* **do**

13   **if** $n_k \in D_+$ **then**

14   Let n_p represent n_k's parent;

15        $\mathbf{W}_k = \frac{|D_+|-1}{|D_+|} \cdot \mathbf{W}_k + \frac{1}{|D_+|} \cdot \mathbf{W}_p - \sum_{n_s \in D_+} \frac{1}{|D_+|} \cdot \mathbf{W}_s$;

// Calculate variances

16 **for** *each node n_k* **do**

17     $\mathrm{Var}(\tilde{f}_k) = \sum_{1 \le j \le |\mathcal{T}|} w_{k,j}^2 \cdot v_j$;

18 **return** $\mathrm{Var}(\tilde{f}_k)$ of each node n_k;

D   ERROR ANALYSIS ON MULTI-DIMENSIONAL QUERIES

Multi-dimensional queries primarily depend on the estimated frequency of 2-D grids, which are adaptively constructed and refined based on the PriPL-Tree for each attribute. As such, we analyze the error in queries on these 2-D grids. As discussed in Section 4.2, for a range query Q that selects a portion r of the area of a grid of size $g_i \times g_j$, the squared error is $2rg_ig_j\sigma^2\binom{m}{2} + r\eta \sum_{c \in G} f_c^2$, where $\sigma^2 = \frac{4e^\epsilon}{N \cdot (e^\epsilon - 1)^2}$. For simplicity, let g represent the maximum grid partition size on each attribute; the asymptotic bound of the squared error is $O(\frac{r \cdot g^2 \cdot m^2}{N \cdot \epsilon^2})$. Furthermore, Theorem D.1 provides the maximum absolute error, derived from Lemma D.2 and the analysis in Section 4.2.

THEOREM D.1. *For any range query Q selecting a portion r of the area of a grid with size g^2, with at least $1 - \beta$ probability,*

$$\max_c |\hat{Q}(r) - Q(r)| = O\left(\frac{rg^2m\sqrt{\log(g^2/\beta)}}{\epsilon\sqrt{N}}\right) \quad (11)$$

LEMMA D.2. *For any cell c in 2-D grids whose frequency is estimated by OUE with $\frac{N}{2 \cdot \binom{m}{2}}$ users, let g denotes the maximum grid partition size on each dimension, $\hat{f}_c$ denote the estimated frequency and f_c denotes the actual frequency, with at least $1 - \beta$ probability,*

$$\max_c |\hat{f}_c - f_c| = O\left(\frac{m\sqrt{\log(g^2/\beta)}}{\epsilon\sqrt{N}}\right) \quad (12)$$

PROOF. For a frequency $\hat{f}_c$ estimated by OUE using $n = \frac{N}{2 \cdot \binom{m}{2}}$ users, it is unbiased and has a variance given by $\mathrm{Var}(\hat{f}_c) = \frac{4e^\epsilon \cdot \binom{m}{2}}{N \cdot (e^\epsilon - 1)^2}$.

By Bernstein's inequality, when ϵ is small,

$$\Pr[|\hat{f}_c - f_c| \ge \delta] \le 2 \cdot \exp\left(-\frac{n^2\delta^2}{2 \cdot n \cdot \frac{4e^\epsilon}{(e^\epsilon-1)^2} + \frac{1}{3} \cdot n \cdot \delta \cdot \frac{2 \cdot e^\epsilon + 1}{e^\epsilon - 1}}\right)$$

$$= 2 \cdot \exp\left(-\frac{n\delta^2}{2 \cdot O(\frac{1}{\epsilon^2}) + \frac{1}{3} \cdot \delta \cdot O(\frac{1}{\epsilon})}\right).$$

By the union bound, there exist $\delta = O\left(\frac{\sqrt{\log(g^2/\beta)}}{\epsilon\sqrt{n}}\right) = O\left(\frac{m\sqrt{\log(g^2/\beta)}}{\epsilon\sqrt{N}}\right)$ such that $\max_c |\hat{f}_c - f_c| \le \delta$ holds with at least $1 - \beta$ probability. □

E   TIME COMPLEXITY ANALYSIS

In this section, we theoretically and experimentally compare the time complexity between PriPL-Tree and its competitors. For convenience, we assume all attributes have the same domain size d.

E.1   1-D Time Complexity

Theoretical Analysis of PriPL-Tree: The construction time follows three phases. Phase 1 is relatively complex, involving the distribution estimation and private PL fitting two steps. The first step mainly includes the user values' aggregation time $O(N \cdot \alpha)$ and the distribution estimation time $O(d \cdot T)$ based on the EM or EMS algorithm within the SW mechansim, where T denotes the number of iterations in EM and EMS. The second step includes matrix-based segment fitting, taking time $O(K^2 \cdot d)$, where calculating the inverse of matrices by the Gaussian elimination method, and interval partitioning, invoking at most $O(K \cdot d)$ segment fitting. When we apply the search acceleration strategy in interval partitioning with a granularity factor ϕ (refer to Section 3.3.2), $O(K \cdot \phi \cdot \log_\phi d)$ times segment fitting is required. Next, in phase 2, involving the construction of the PriPL-Tree and the estimation of each node's frequency, the time complexity is $O(K + N \cdot (1 - \alpha) \cdot K)$. Phase 3, refining the PriPL-Tree through two traversals, leads to a complexity of $O(K)$. Overall, the construction time complexity primarily stems from private user data aggregation, frequency estimation, and private PL fitting during phases 1 and 2, totaling $O(N \cdot K + d \cdot T + d \cdot \log d \cdot K^3)$. This has been confirmed across various datasets and domain sizes d,

as illustrated in Figure 17. The legend labeled "LDP frequency estimation" represents user data aggregation and frequency estimation using existing LDP mechanisms.

(a) Varying Datasets with $d = 1024$
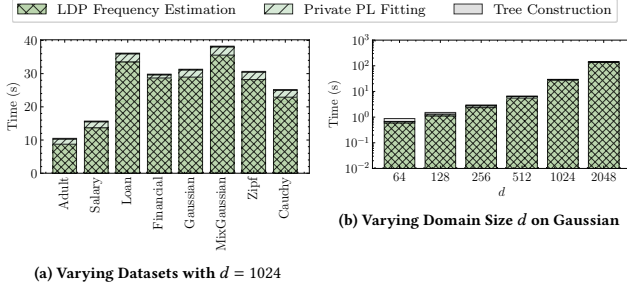
(b) Varying Domain Size d on Gaussian

Figure 17: Construction Time of PriPL-Tree

Theoretical Analysis of Competitors: In Table 3, we compare the construction and query time complexities of different methods in 1-D scenarios. Because all compared methods utilize hierarchical tree structures, the construction time complexity of them primarily stems from processing users' reports, while the query time complexity depends on the tree height. In this table, the construction time complexities for AHEAD and DHT are sourced from [7], and that for PrivNUD is analyzed by us using the same methodology. Since that the maximum segment number K in PriPL-Tree is typically small, our construction time complexity is competitive with others, and our query time is significantly lower than our competitors.

Table 3: Time Complexity Comparison for 1-D Range Queries

Methods	Construction Time	Query Time
PriPL-Tree	$O(N \cdot K + d \cdot T + d \cdot \log d \cdot K^3)$	$O(\log_2 K)$
PrivNUD	$O(\log_d 2 \cdot N \cdot d)$	$O(\log_2 d)$
AHEAD	$O(\log_d 2 \cdot N \cdot d)$	$O(\log_2 d)$
DHT	$O(N + d^3)$	$O(\log_2 d)$

Experimental Comparison: To ensure a fair comparison of method runtimes, we reimplemented DHT, originally in C++, in Python to match the programming language of the other methods. We have analyzed the construction and query times across different datasets in Section 5.2. Additionally, we evaluate the construction times of different methods on varying domain sizes d and the query times on varying query volumes.

For construction time, all methods are mainly influenced by the time of aggregating users' perturbed messages, a step inherent in employed LDP frequency estimation mechanisms, and thus increase with domain size d. In our comparisons, AHEAD [7] and PrivNUD [37] share the same asymptotic time complexity in Table 3, but show significant discrepancies in experimental performance. This variation stems from their different implementations, where AHEAD utilizes the Treelib package for tree structures, similar to ours, while PrivNUD uses a nested array.

For query time, DHT shows a rising trend with increasing query volume $Vol(Q)$, whereas others, including PriPL-Tree, AHEAD, and PrivNUD, initially rise then decline as query volume increases. This

trend is significantly pronounced for PrivNUD but slight for PriPL-Tree and AHEAD. This observation aligns with the theoretical analysis that in the hierarchical trees, the query time correlates with the differing numbers of nodes accessed at various query volumes, typically peaking when the query volume is around 50%.
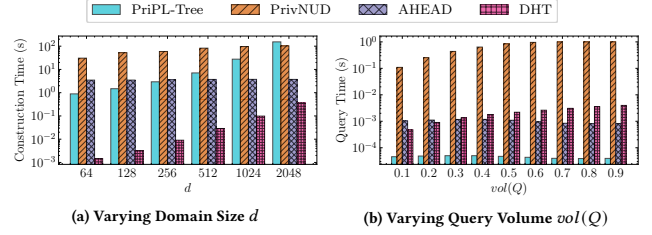

(a) Varying Domain Size d

(b) Varying Query Volume $vol(Q)$

Figure 18: Runtime Evaluation on Gaussian

E.2 Multi-D Time Complexity

Theoretical Analysis: In multi-dimensional scenarios, leveraging the 1-D and 2-D estimations is a common strategy across all methods. The construction time complexity of these hybrid low-dimensional data structures is primarily from processing N user's reports, where each user reports multiple perturbed counts for nodes in the tree or cells in the grid. The query time complexity arises from computing the frequencies of the corresponding $\binom{\lambda}{2}$ 2-D grids and generating the response matrix with 2^λ values. Because generating the response matrix with given $\binom{\lambda}{2}$ 2-D grids is uniform for all methods, we focus on the time complexity of 2-D grids estimation for comparison, omitting the time consumption of the response matrix step. Following this idea, we analyze the time complexity of our PriPL-Tree and other competitors in Table 4. During our analysis, we note that the granularity of the grids in the PriPL-Tree method is typically smaller than the maximum number of segments per attribute, denoted as K; therefore, we set the granularity to $O(K)$. For other methods, we use the optimized granularity reported in their respective papers.

Table 4: Time Complexity Comparison for λ-d Range Queries on m-d Datasets

Methods	Construction Time	Query Time
PriPL-Tree	$O(N \cdot K^2 + m \cdot d \cdot T)$	$O(\lambda^2 \cdot K^2)$
PrivNUD	$O(\log_d 2 \cdot N \cdot d/m + N\sqrt{N}/m)$	$O(\lambda^2 \cdot \sqrt{N}/m)$
AHEAD	$O(\log_d 4 \cdot N \cdot d^2)$	$O(\lambda^2 \cdot \log_4 d^2)$
HDG	$O(N\sqrt{N}/m)$	$O(\lambda^2 \cdot \sqrt{N}/m)$
PRISM	$O(N\sqrt{N}/m)$	$O(\lambda^2 \cdot \sqrt{N}/m)$

∗: K is the maximum segment number in PriPL-Tree, and T is the number of iterations in the EM algorithm within the SM protocol [25].

Experimental Comparison: In Figure 19, we assess the actual runtime of these methods. For construction time across eight different datasets, i.e., Figure 19 (a), our PriPL-Tree performs comparably to PrivNUD and is slightly longer than HDG, which is the fastest and only uses grid structures without trees. In Figure 19 (c), the construction times for all methods increase with data dimensionality.

PRISM is an exception, with significantly larger time consumption in 2-D cases. This is because it generates significantly finer grids in the 2-D case than in other dimensions with its granularity optimization strategy [41]. For query time, i.e., Figure 19 (b) and (d), our PriPL-Tree method consistently achieves the fastest responses, with times increasing as query dimension λ increases.
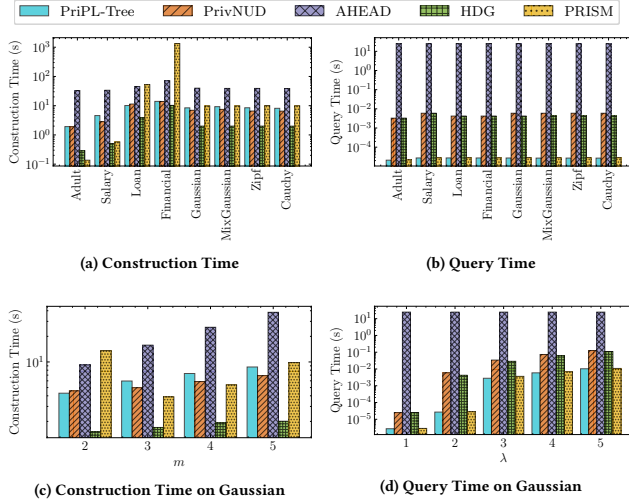

(a) Construction Time **(b) Query Time**

(c) Construction Time on Gaussian **(d) Query Time on Gaussian**

Figure 19: Runtime Evaluations (Defaulty, $m = 5$ and $\lambda = 2$)

F DATASETS DESCRIPTION

In this section, we provide additional descriptions of both synthetic and real-world datasets. For our main 1-D scenario, we illustrate the distribution for the default attribute of each dataset in Figure 20. For multi-dimensional scenarios, we present the mean, variance, skewness, and kurtosis for each attribute across all datasets. Skewness indicates data distribution asymmetry, with positive values suggesting a long right tail and negative values a long left tail. Kurtosis measures the peak sharpness and tail thickness of the distribution. We bold the Kurtosis values exceeding 3 in the following tables, indicating an excessively peaked distribution. Given the similar statistical characteristics across dimensions, we summarize the average indicators for synthetic datasets in Table 5. For real-world datasets, we detail these characteristics per attribute in the following: Adult dataset in Table 6, Loan dataset in Table 7, Salary dataset in Table 8, and Financial dataset in Table 9.

Table 5: Summary of Synthetic Datasets ($d = 256$)

Dataset	Mean	Variance	Skewness	Kurtosis
Gaussian	155.0	775.34	1.44	-0.28
MixGaussian	135.12	2516.62	1.15	-0.28
Cauchy	159.37	609.50	4.42	**15.46**
Zipf	24.40	2517.47	18.58	**284.47**

(a) Gaussian **(b) MixGaussian**

(c) Cauchy **(d) Zipf**

(e) Adult ("fnlwgt") **(f) Loan ("total_pymnt")**

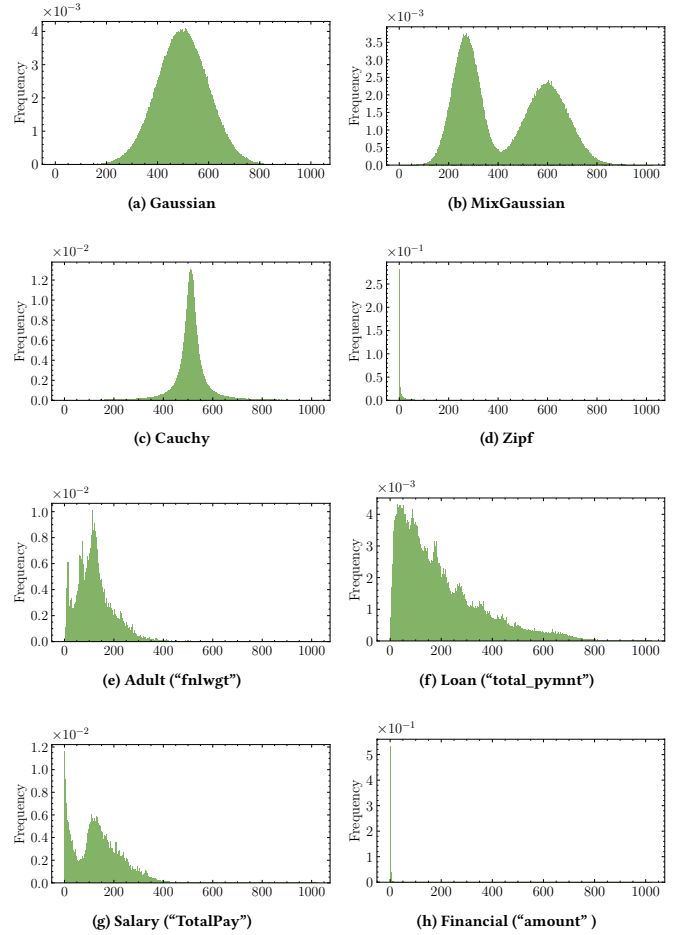**(g) Salary ("TotalPay")** **(h) Financial ("amount")**

Figure 20: Data Distributions of 1-D Datasets

Table 6: Summary of Adult Dataset ($d = 256$)

Attribute	Mean	Variance	Skewness	Kurtosis
fnlwgt	30.36	336.88	2.19	**4.03**
age	21.58	186.06	-0.03	-1.64
capital-gain	2.71	353.74	15.90	**250.75**
capital-loss	5.11	555.48	15.90	**250.93**
hours-per-week	39.44	152.45	8.82	**80.72**

G ADDITIONAL EVALUATION ON HIGH-DIMENSIONAL DATASET

In this section, we evaluate performance on the high-dimensional IPUMS dataset, as shown in Figure 21. This dataset, sourced from the 2022 IPUMS repository [1], contains 50 dimensions and 15,721,123 samples. Due to the high runtime complexity of AHEAD, as detailed in Appendix E.2, it could not produce effective results on 50-D datasets within the limited time in our experimental setup, so we excluded its results from the figure. This omission does not impact
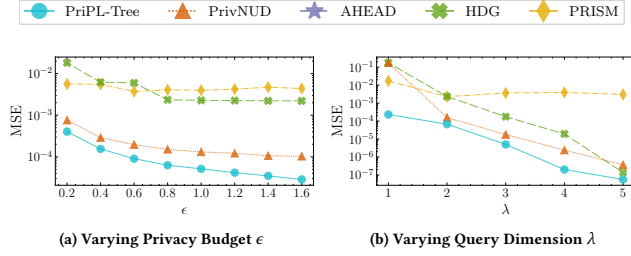
---
[1] https://usa.ipums.org/usa/

(a) Varying Privacy Budget ϵ (b) Varying Query Dimension λ

Figure 21: Evaluation of Range Queries on 50-D IPUMS

Table 7: Summary of Loan Dataset ($d = 256$)

Attribute	Mean	Variance	Skewness	Kurtosis
total_pymnt	48.67	1620.19	1.27	0.35
total_rec_int	21.69	596.95	2.98	**8.38**
installment	66.01	1595.08	1.12	0.37
total_il_high_credit_limit	5.02	29.44	5.61	**32.28**
loan_amnt	93.03	3661.59	4.75	**26.27**

Table 8: Summary of Salary Dataset ($d = 256$)

Attribute	Mean	Variance	Skewness	Kurtosis
TotalPay	33.59	515.88	1.80	2.56
TotalPayBenefits	42.15	797.60	1.64	2.64
BasePay	52.80	1174.51	1.46	2.33
OvertimePay	5.08	140.77	15.43	**240.57**
OtherPay	6.22	25.47	13.47	**194.04**

Table 9: Summary of Financial Dataset ($d = 256$)

Attribute	Mean	Variance	Skewness	Kurtosis
amount	0.23	2.65	15.76	**247.68**
oldbalanceOrg	3.40	151.81	15.82	**249.22**
newbalanceOrig	4.24	225.10	15.72	**246.97**
oldbalanceDest	0.57	5.61	15.57	**243.49**
newbalanceDest	0.64	6.60	15.47	**241.30**

our analysis, as previous experiments have already demonstrated its inferior performance compared to PrivNUD and our PriPL-Tree.

Overall, PriPL-Tree consistently achieves the lowest MSE among all competitors. For various privacy budgets (Figure 21 (a)), PriPL-Tree shows a reduction in MSE ranging from 46.34% to 72.22%, with an average reduction of 60.67%. Across different query dimensions (Figure 21 (b)), it reduces MSE by 56.80% to 91.72%, averaging 69.90%. The superiority of PriPL-Tree here is more pronounced than in experiments on only 5-D datasets, as shown in Figures 9 and 10.

Furthermore, Figure 21 (b) shows a decrease in MSE with increasing query dimension λ, in contrast to the results from 5-D synthetic Gaussian datasets depicted in Figure 10 (b), where MSE increases with λ. This discrepancy arises from the different characteristics of the datasets. Methods estimating high-dimensional range queries using low-dimensional responses are more accurate when attribute correlations are minor. The Gaussian dataset, with high inter-attribute correlation (covariance of 0.6), shows decreasing accuracy as λ increases. In contrast, IPUMS, with generally lower correlations among attributes, allows for relatively precise estimates across varying dimensions. The observed decrease in MSE is primarily due to the lower actual frequency of higher-dimensional queries, where the error is proportional to this frequency.