# Collecting and Analyzing Data Jointly from Multiple Services under Local Differential Privacy

Min Xu
University of Chicago
xum@cs.uchicago.edu

Bolin Ding
Alibaba Group
bolin.ding@alibaba-inc.com

Tianhao Wang
Purdue University
tianhaowang@purdue.edu

Jingren Zhou
Alibaba Group
jingren.zhou@alibaba-inc.com

## ABSTRACT

Users' sensitive data can be collected and analyzed under local differential privacy (LDP) without the need to trust the data collector. Most previous work on LDP can be applied when each user's data is generated and collected from a single service or data source. In a more general and practical setting, sensitive data of each user needs to be collected under LDP from multiple services independently and can be joined on, *e.g.*, user id. In this paper, we address two challenges in this setting: first, how to prevent the privacy guarantee from being weakened during the joint data collection; second, how to analyze perturbed data jointly from different services. We introduce the notation of user-level LDP to formalize and protect the privacy of a user when her joined data tuples are released. We propose mechanisms and estimation methods to process multi-dimensional analytical queries, each with sensitive attributes (in its aggregation and predicates) collected and perturbed independently in multiple services. We also introduce an online utility optimization technique for multi-dimensional range predicates, based on consistency in domain hierarchy. We conduct extensive evaluations to verify our theoretical results using synthetic and real datasets.

**PVLDB Reference Format:**
Min Xu, Bolin Ding, Tianhao Wang, Jingren Zhou. Collecting and Analyzing Data Jointly from Multiple Services under Local Differential Privacy. *PVLDB*, xx(xxx): xxxx-yyyy, 2020.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

## 1. INTRODUCTION

Sensitive data, or *attributes,* about users' profiles and activities is collected by enterprises and exchanged between different services in one organization to help make informed data-driven decisions. The *de facto* privacy standard, differential privacy (DP) [13], has been deployed in several scenarios to provide rigorous privacy guarantees on how attributes are collected, managed, and analyzed. Informally, differential privacy requires that the output of data processing varies little with any change in an individuals' attributes.

The centralized model of DP assumes that a *trusted party* collects exact attribute values from users. The trusted party is inside a physical or logical privacy boundary [21], and injects noise during the (offline or online) analytical process so that query results transferred across the firewall ensure DP. Systems along this line [24, 25, 18, 21, 20] extend the class of data schemes and queries supported with improving utility-privacy trade-off. A use case is that a data engine managing customers' sensitive data, *e.g.*, in Uber [18], provides a query interface satisfying DP for its employees.

In the absence of the central trusted party, the *local differential privacy* model (LDP) [12] is adopted. Each user has her attribute values locally perturbed by a randomized algorithm with LDP guaranteed, *i.e.*, the likelihood of any specific output of the algorithm varies little with input; each user can then have the perturbed values leave her device, without the need to trust the data collector. Analytical queries can be answered approximately upon a collection of LDP perturbed values. Apple [1], Google [15], and Microsoft [9] deploy LDP solutions in their applications on users' device to estimate statistics about user data, *e.g.*, histograms and means.

In this paper, we investigate how to collect and analyze multi-dimensional data under LDP in a more general setting: each user's attributes are collected by multiple independent services, with LDP guaranteed; and data tuples from the same user collected across different services can be joined on, *e.g.*, user id or device id which is typically known to the service providers. Two natural but open questions are: *what privacy guarantee can be provided on the joined tuples for each user*, and *what analytics can be done on the jointly collected (and independently perturbed) multi-dimensional data.*

**Data model for joint collection and analytics.** Figure 1 illustrates a target scenario. Two users $u$ and $u'$ are active in service 1 and service 2. Attribute tuples $T_1^u$ and $T_1^{u'}$ are collected in service 1 from $u$ and $u'$, respectively; and it is similar for service 2. Indeed, each tuple is perturbed by a randomized algorithm $\mathcal{R}$ to ensure LDP. Conceptually, $T_1$ (or $T_2$) is a relational table holding users' information collected from service 1 (or service 2); and perturbed versions of $T_1$ and $T_2$ can be joined on user id, which is naturally known to both service providers. The questions are: whether it is safe to release the joined tuples in $\mathcal{R}(T_1) \bowtie \mathcal{R}(T_2)$, and what analytics can be conducted on it. Here is a motivating example.

**Example 1.1** (E-Commerce)**.** A user profiling service and a transaction processing service collect separate information about users in a E-commerce platform. The user profiling service collects attributes, including Age and Income, into table User. The transaction processing service collects the attributes about transactions, including Amount and Category, into table Transaction. Each user has a unique UID, which is randomly generated at sign-up, across the two services. An analyst wants to know the average amount on sports products for users in specific age group, *e.g.*,

```
SELECT AVG(Amount) FROM
Transaction JOIN User ON User.UID = Transaction.UID
WHERE Category = Sports AND Age ∈ [20, 40].
```

*Proceedings of the VLDB Endowment,* Vol. xx, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

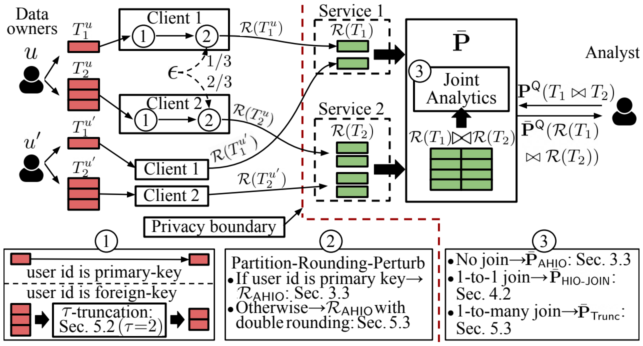

Data owners
$u$
$u'$
$T_1^u$
$T_2^u$
$T_1^{u'}$
$T_2^{u'}$
Client 1
Client 2
Client 1
Client 2
1
2
$\epsilon$
1/3
2/3
$\mathcal{R}(T_1^u)$
$\mathcal{R}(T_2^u)$
$\mathcal{R}(T_1^{u'})$
$\mathcal{R}(T_2^{u'})$
Service 1
$\mathcal{R}(T_1)$
Service 2
$\mathcal{R}(T_2)$
$\bar{\mathbf{P}}$
3
Joint Analytics
$\mathcal{R}(T_1) \bowtie \mathcal{R}(T_2)$
Analyst
$\mathbf{P}^{\mathsf{Q}}(T_1 \bowtie T_2)$
$\bar{\mathbf{P}}^{\mathsf{Q}}(\mathcal{R}(T_1) \bowtie \mathcal{R}(T_2))$
Privacy boundary
1
user id is primary-key
user id is foreign-key
$\tau$-truncation: Sec. 5.2 ($\tau$=2)
2
Partition-Rounding-Perturb
• If user id is primary key→ $\mathcal{R}_{\mathsf{AHIO}}$: Sec. 3.3
• Otherwise→$\mathcal{R}_{\mathsf{AHIO}}$ with double rounding: Sec. 5.3
3
• No join→$\bar{\mathbf{P}}_{\mathsf{AHIO}}$: Sec. 3.3
• 1-to-1 join→$\bar{\mathbf{P}}_{\mathsf{HIO\text{-}JOIN}}$: Sec. 4.2
• 1-to-many join→$\bar{\mathbf{P}}_{\mathsf{Trunc}}$: Sec. 5.3

Figure 1: Data collection and analytics across multiple services (top), and an algorithmic pipeline of our solution (bottom).

The goal of this paper is *to support a class of* multi-dimensional analytical *queries against joins of tuples with attributes collected via multiple services from data owners*. A query here aggregates (COUNT, SUM, or AVG) on attributes of tuples under point and range predicates on the attributes (in the WHERE clause)—all attributes in the aggregation and predicates could be sensitive and come from different services. Meanwhile, we want to provide strengthened privacy guarantee (*i.e.*, *user-level local differential privacy* as introduced later) for every single user given that her tuples collected from different services can be linked together with her user id (known to service providers).

**Challenge I (joint aggregation).** The setting and the query class studied in this paper are much broader than previous works in two important aspects. **1)** We allow all attributes (except user ids which are naturally known to service providers) in the multi-dimensional analytical queries to be sensitive, while existing works on answering range queries [32, 8] and releasing marginals [7, 36, 28] under LDP cannot handle aggregations on sensitive attributes. Note that it is easy to handle non-sensitive attributes by simply plugging their true values when evaluating aggregations or predicates as in [32]. **2)** Most previous works on multi-dimensional analytics, *e.g.*, [32] and [36], collect and analyze data from a single service only; in our setting, attributes in a query may come from different services, each of which perturbs its sensitive attributes independently—existing methods only work when these attributes are perturbed as a whole, which is impossible if these attributes are from different services.

**Challenge II (frequency-based attack).** We want to point out that it is *insufficient* (in terms of privacy) to ensure LDP independently in each service, given the fact that tuples collected in different services can be joined on user id which is known to service providers.

A straw-man solution based on $\epsilon$-LDP works as follows: at data collection, divide the privacy budget $\epsilon$ among the tuples from different services for each user, and perturb each tuple using an LDP mechanism with proper budget. The hope is that, based on the sequential composition, the overall privacy guarantees for each user is $\epsilon$-LDP. However, it turns out that this solution cannot provide any differential privacy for a user because it releases the sensitive information, the numbers of tuples that a user generates while using a service, after tuples are joined on user id. Such exact frequency release enables various attacks to users, depending on the prior knowledge of the adversary. For instance, as shown in Figure 2 (an instance of Example 1.1), if the adversary has the prior knowledge that the Income of a user is equal to the number of transactions multiplied by 100, then having access to the exact number of transactions of a user (even though each transaction is perturbed under LDP) enables the recovery of the sensitive Income of the user.

The above frequency-based attack is particularly feasible in our target scenario because we assume that the same user id is attached to tuples collected from a user, and is accessible by the service providers. User id is not a sensitive attribute, but can function as a join key and allows the adversary to group tuples of the same user.

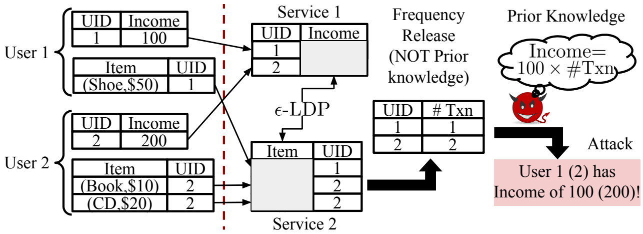


Figure 2: Frequency-based attack (# Txn = number of transactions).

### Contributions and Solution Overview

Our main contribution is to extend the setting and query class supported by existing LDP mechanisms from single-service frequency queries to the more complex ones, including aggregations on sensitive attributes and multi-service joint aggregation; as we point out in challenges I-II, no existing LDP mechanisms can handle our target setting and query class with formal privacy and utility guarantees. We first give an overview of important algorithmic components in our solution and introduce the end-to-end pipeline.

• *Partition-rounding-perturb framework.* We propose a framework to extend the class of multi-dimensional analytical queries with aggregations on sensitive attributes, which is left as an open problem in [32]. The main idea is to first randomly round the value of an attribute to the two ends, with rounding probability carefully chosen so that the aggregation on the rounded variable is an unbiased estimation of the true aggregation of the attribute. The rounded variable with two possible values is treated as a new sentitive attribute of the user and then fed into the local perturbation mechanism for LDP, *e.g.*, the one in [32]. We note that the rounding may incur some utility loss, depending on how large attribute domain is, but overall, the loss is still dominated by the part incurred by the local perturbation. Moreover, users are randomly partitioned to boost the utilty when there are multiple attributes to be aggregated.

• *Independent perturbation and joint analytics (*HIO-JOIN*).* In our solution, tuples collected from different services are perturbed independently, and we need new estimation algorithm to estimate aggregations on the joins of these tuples. Our solution HIO-JOIN generalizes the split-and-conjunction technique and the mechanism HIO based on hierarchical intervals [32] to estimate the joint distribution on the vector of perturbed tuples and evaluate how likely the joint predicate is true. To improve the utility, we show that the standard trick of randomly partitioning users [8] also works here. In addition, we propose an utility optimization technique, by enforcing consistency across different levels in the hierarchical intervals.

• *User-level LDP and $\tau$-truncation.* To tackle the frequency-based vulnerability for rigorous privacy guarantees, we formally define *user-level local differential privacy* (uLDP), which gathers all the information about a user (after the join) and makes it indistinguishable as whole just as in the classical LDP notation. For one-to-one joins, the above innovations suffice to ensure uLDP. However, in one-to-many joins, a user can generate one or more tuples in a particular service; these tuples can be joined with tuples generated by her in other services. How many tuples are joinable for her (*i.e.*, *frequency*) is part of the output of the data collection mechanism, and thus needs to be hidden (otherwise, uLDP is violated).

We propose the $\tau$-truncation technique to hide such frequency information. Informally, no matter how many tuples are joinable for a user, we randomly sample (or copy) $\tau$ of them and feed them into the perturbation step. Each sample tuple is associated with a weight (which is inversely proportional to the sampling ratio) to compensate for the contribution to the aggregation from tuples not in the sample. This weight is a new sensitive attribute (of the user), as it depends on the true frequency. A technique called *double rounding* is proposed to perturb these weights together with other attributes and to derive unbiased estimation of the aggregation.

**Overview of Solution Pipeline.** Our solution requires minimum coordination between different services. The whole pipeline of data collection and analytics is described in Figure 1 (bottom part).

For each user and each service, ①: if user id is the primary key of tuples generated from this service (*e.g.*, user profiling service), one tuple is to be collected per user; if user id is a foreign key of tuples from this service (*e.g.*, transaction service), the number of tuples per user is uncertain. In the latter case, we use our $\tau$-truncation technique to randomly sample (or create) $\tau$ tuples for each user and collect them, even if zero tuple is generated from this service (meaning that the user has not used this service), in order to hide the frequency/existence information about this user in the service. Then, depending on how many tuples in total are to be collected from all the services after $\tau$-truncation (*e.g.*, in Figure 1, 3 tuples are collected per user in total), our privacy budget is evenly divided, and ②: each tuple from each service is perturbed independently using our partition-rounding-perturb framework before collected to enable both frequency and attribute aggregations (if the tuple contains user id as the primary key, then a single rounding on an attribute is applied; otherwise, double rounding is needed).

Note that the data collection is *query-independent*—we do not have to know in advance which attributes will appear in the query and where they are. The only information that needs to be coordinated between services is the value of $\tau$ (which will be specified later) and the total number of tuples to be collected for each user.

To process a multi-dimensional aggregation query, tuples (with attributes in the query) collected from different services are joined on user id, and ③: the query is rewritten into frequency queries on their joint distribution, which are combined into the query result: if the query only contains attributes in a single table, *i.e.*, no join, we apply the estimation of partition-rounding-perturb, plugged with that of HIO; if it contains attributes across multiple tables, all with user id as primary key, *i.e.*, 1-to-1, we apply the estimation of partition-rounding-perturb, plugged with that of HIO-JOIN; and if it contains attributes across multiple tables, with user id as foreign key in some table, *i.e.*, 1-to-many, we apply the estimation of partition-rounding-perturb, plugged with those of HIO-JOIN and double rounding.

**Organization.** We introduce our target data model, the basic LDP definition and notations in Section 2. We further extend the basic LDP to the user-level privacy guarantee, *i.e.*, uLDP, in Section 2.3. Section 3 focuses on handling aggregations of sensitive attributes. Section 4 focuses on joint aggregation with one-to-one relation. We address the frequency-based attack in joint aggregation with one-to-many relation in Section 5. Section 6 introduces our utility optimization technique. Experimental results are reported in Section 7. Extensions and more related work are discussed in Sections 8-9.

# 2. PRELIMINARIES

## 2.1 Data Model and Analysis Tasks

We assume that there are $n$ users and $K$ services. Each service collects users' data $t$, in the form of tuple(s) of *attributes*, with a client-side application. Some service collects exactly one tuple from each user. Other services may collect multiple tuples (*e.g.*, transactions). We denote the tuples collected by the $i$-th service as a relational table $T_i$. And we denote all the tuples collected from user $u$ as $T^u$, and those from $u$ collected by the $i$-th service as $T_i^u$. Thus, $\forall u \in [n], \bigcup_{i=1}^{K} T_i^u = T^u$, and $\forall i \in [K], \bigcup_{u=1}^{n} T_i^u = T_i$.

Besides the attributes about the users, we assume that each tuple has a pseudo-random user id (UID), which is known to the service provider and can serve as the non-sensitive join key. An analyst can jointly analyze the data across multiple services $S \subseteq [K]$ by joining their tuples, on the user id, as a *fact table* $\Join_{i \in S} T_i$.

In this paper, we focus on answering *multi-dimensional analytical* (MDA) queries against joins of tables. Consider a function F that takes one of the form COUNT, SUM, or AVG on an attribute $A$. A multi-dimensional analytical query takes the format:

SELECT F($A$) FROM $T_{i_1}$ JOIN … JOIN $T_{i_q}$ ON UID WHERE C,

where $\{T_{i_1}, \dots, T_{i_q}\}$ are the tables to be joined on UID; C is the predicate on attributes of the joined fact table, and it can be conjunction of either point constraints for categorical attributes, or range constraints for ordinal attributes, or their combinations. We focus on conjunctions of constraints in this paper. Disjunctions can be handled by combinations of conjunctions, as described in [32].

Since user id is known to service providers, other attributes may leak sensitive information about each user. In the next two subsections, we will introduce the classical model of *local differential privacy*, and propose an enhanced version, called *user-level local differential privacy* (for the setting with multiple services), to protect these sensitive attributes during data collection. In the rest part of this paper, for the ease of description, we assume that all the attributes in a query are sensitive. We can extend our techniques for queries with non-sensitive attributes by simply plugging their true values when evaluating aggregations or predicates as in [32].

## 2.2 Local Differential Privacy

Local differential privacy (LDP) can be used for the setting where each user possesses one value $t$ from a fixed domain $\mathbb{D}$. We first review this notation, and will extend it for the setting of our paper.

Consider a randomized algorithm $\mathcal{R}(t)$ that takes $t \in \mathbb{D}$. The formal definition of privacy of $\mathcal{R}(t)$ is defined as follows:

**Definition 1** ($\epsilon$-Local Differential Privacy)**.** A randomized algorithm $\mathcal{R}$ over $\mathbb{D}$ satisfies $\epsilon$-local differential privacy ($\epsilon$-LDP), where $\epsilon \geqslant 0$, if and only if for any input $t \neq t' \in \mathbb{D}$, we have

$$\forall y \in \mathcal{R}(\mathbb{D}): \ \mathbf{Pr}\left[\mathcal{R}(t) = y\right] \leqslant e^{\epsilon} \cdot \mathbf{Pr}\left[\mathcal{R}(t') = y\right],$$

where $\mathcal{R}(\mathbb{D})$ denotes the set of all possible outputs of $\mathcal{R}$.

In the definition, $\epsilon$ is also called the privacy budget. A smaller $\epsilon$ implies stronger privacy guarantee for the user, as it is harder for the adversary to distinguish between $t$ and $t'$. Since a user never reveals $t$ to the service but only reports $\mathcal{R}(t)$, the user's privacy is still protected even if the service is malicious.

**Sequential composability.** An important property, sequential composability [24], for LDP bounds the privacy on $t$ when it is perturbed multiple times. We state the property in Proposition 2.

**Proposition 2** (Directly from [24])**.** Suppose $\mathcal{R}_i$ satisfies $\epsilon_i$-LDP, the algorithm $\mathcal{R}$ which releases the result of each $\mathcal{R}_i$ on input $t$, *i.e.*, $\mathcal{R}(t) = \langle \mathcal{R}_i(t) \rangle_{i=1}^{k}$, satisfies $\sum_{i=1}^{k} \epsilon_i$-LDP.

## 2.3 User-level Privacy across Multiple Services

In our setting, multiple services collect tuples from users and thus sensitive information comes from separate domains. For instance, in Example 1.1, one service collects users' profile, while the other collects users' online shopping transactions. Definition 1 cannot quantify the privacy for this setting. Note that [9] studies LDP in an industrial deployment with multiple services, each of which collects one telemetry data. Here, we define the privacy guarantee formally in a more general setting, where a user can generate arbitrary number of tuples when using a service. In addition, we assume tuples of a user are collected only once, which is orthogonal to the continuous observation model [19].

| Symbol | Definition |
|---|---|
| $n$, $K$ | The number of users and services, respectively |
| $[n]$ | The set of integers $\{1, \dots, n\}$ |
| $t \in \mathbb{D}$ | A tuple in the domain $\mathbb{D}$ |
| $T_i, T^u, T_i^u$ | Tuples generated in service $i \in [K]$, by user $u \in [n]$ and by user $u$ in service $i$ |
| $\mathsf{Q}$, $\mathsf{F}$, $\mathsf{C}$ | Query, aggregation function and predicate |
| $d$, $d_q$ | The numbers of attributes in $T_i$ ($i \in [K]$) and $\mathsf{C}$ |
| $A$ | Attribute with domain of range $[A_{\min}, A_{\max}]$ |
| $m$ | The cardinality of the domain of an attribute |
| $y \leftarrow \mathcal{R}(t)$ | Perturb $t$ into $y$ using algorithm $\mathcal{R}$ under $\epsilon$-LDP |
| $\mathcal{S}$, $\mathcal{M}$ | The attribute partition and rounding function |
| $\tau, \tau_{\max}$ | The truncated and the max numbers of tuples a user generates in a service |
| $\mathbf{P}^{\mathsf{Q}}(\bar{\mathbf{P}}^{\mathsf{Q}})$ | The true (estimated) aggregation for query $\mathsf{Q}$ |
| $\mathbf{f}_{\mathsf{C}}$ ($\mathbf{f}'_{\mathsf{C}}$) | Vector of frequencies of the true (perturbed) tuples for $2^{d_q}$ true/false combinations of conditions in $\mathsf{C}$ |
| $\mathbf{M}$ | The state transition matrix |
| $r$ | Truncation weight of a tuple in $\tau$-truncation |
| $B, I, \mathcal{I}$ | Hierarchical fan-out, interval, and decomposition |

Figure 3: Table of notations

In this paper, we consider the worst case where different services come together to infer the user's sensitive information, and we aim to protect the privacy of a user over the joint domain for all possible tuples across the $K$ services. Suppose the $i$-th service collects tuples from the domain $\mathbb{D}_i$. A user $u$ may generate zero, one, or multiple tuples when using the $i$-th service, denoted by a multiset $T_i^u \subseteq \mathbb{D}_i$. Across the $K$ services, the user generates tuples $\langle T_1^u, \dots, T_K^u \rangle$ in the joint domain. We define the user-level privacy spanning multiple services below:

**Definition 3** ($\epsilon$-uLDP)**.** A randomized algorithm $\mathcal{R}$ over the joint domain across $K$ services is user-level $\epsilon$-locally differentially private ($\epsilon$-uLDP), if and only if for any two users $u, u'$, and their collected tuples $T^u = \langle T_1^u, \dots, T_K^u \rangle$ and $T^{u'} = \langle T_1^{u'}, \dots, T_K^{u'} \rangle$ in the joint domain s.t. $\exists i \in [K]$: $T_i^u \neq T_i^{u'}$, we have:

$$\forall y \in \mathcal{R}(\mathbb{D}_{1,\dots,K}):\ \mathbf{Pr}\left[\mathcal{R}(T^u) = y\right] \leqslant e^{\epsilon} \cdot \mathbf{Pr}\left[\mathcal{R}(T^{u'}) = y\right],$$

where $\mathcal{R}(\mathbb{D}_{1,\dots,K})$ is the output domain across the $K$ services.

When $K > 1$ and $\forall u, u', |T_i^u| = |T_i^{u'}|$, all users have the same number of tuples collected by each service, and the only privacy loss is from the joint distribution of values of the user's tuples, which we can bound using the basic LDP, plus sequential composition. And when $K > 1$ and $\exists u, u', s.t., |T_i^u| \neq |T_i^{u'}|$, uLDP covers the more general setting where a user can generate arbitrary number of tuples for a service, which is the focus of our work.

$\epsilon$-uLDP is a general privacy definition for the multi-service data collection setting, and it covers the frequency-based attacks described in Section 1: for users $u, u'$, with different numbers of tuples on the $i$-th service, *i.e.*, $|T_i^u| \neq |T_i^{u'}|$, the straw-man mechanism, *i.e.*, splitting $\epsilon$ among $T^u$ ($T^{u'}$), and perturbing each of them with LDP, provides no uLDP guarantees. That is, for $y$ with $|T_i^u|$ perturbed values for the $i$-th service, where $i \in [K]$, the straw-man has $\mathbf{Pr}[\mathcal{R}(T^u) = y]$ being non-zero while $\mathbf{Pr}[\mathcal{R}(T^{u'}) = y]$ being zero, which implies $\infty$-uLDP, or no privacy.

In Section 3, 4, we mainly focus on the utility for our target query class, and assume each service collects exactly one tuple from each user. In Section 5, we propose mechanism that achieves $\epsilon$-uLDP and prevents the frequency-based attacks in the general setting.

**Notations.** For the convenience of the reader, we summarize important notations introduced throughout the paper in Figure 3.

# 3. ATTRIBUTE AGGREGATION

To handle multi-dimensional analytical queries under LDP, existing works propose multi-dimensional *frequency oracles* over single tables. In this section, we first review two such LDP oracles. In order to estimate aggregations on sensitive attributes, we introduce a new class of *sensitive-weight frequency oracles* based on *stochastic rounding* and a new framework called *partition-rounding-perturb*, which allows the same sensitive attribute to appear in both aggregations and predicates of the queries. Two instantiations of this framework are introduced, with provable error bounds.

## 3.1 Building Block: Frequency Oracles

A multi-dimensional frequency oracle under LDP is a pair of algorithms, $(\mathcal{R}, \bar{\mathbf{P}})$: $\mathcal{R}$ follows LDP definition (Definition 1); and $\bar{\mathbf{P}}^{\mathsf{C}}(y)$ is a deterministic algorithm that takes $y$, *i.e.*, the output of $\mathcal{R}$, and outputs its estimated contribution to the predicate $\mathsf{C} \subseteq \mathbb{D}$. In addition, we denote the true result of the original tuple as $\mathbf{P}^{\mathsf{C}}(t)$.

For a fact table $T$ of $n$ tuples, we denote $\bar{\mathbf{P}}^{\mathsf{C}}(\mathcal{R}(T)) = \sum_{y \in \mathcal{R}(T)} \bar{\mathbf{P}}^{\mathsf{C}}(y)$ as the estimated frequency aggregation of $\mathsf{C}$ against $T$, and $\mathbf{P}^{\mathsf{C}}(T)$ as the true frequency. And we evaluate the utility using *mean squared error* (MSE), over the randomness in $\mathcal{R}$:

$$\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{C}}(\mathcal{R}(T))) = \mathbf{E}[(\bar{\mathbf{P}}^{\mathsf{C}}(\mathcal{R}(T)) - \mathbf{P}^{\mathsf{C}}(T))^2].$$

### 3.1.1 Optimal Local Hashing

Optimal local hashing ($\mathsf{OLH}$) [31] uses hash function $H : \mathbb{D} \mapsto [g]$ to compress the domain $\mathbb{D}$. Here, $H$ is randomly selected from a pre-defined family of hash functions, whose $g$ is the closest integer to $e^{\epsilon} + 1$. Given the tuple $t$, $\mathcal{R}_{\mathsf{OLH}}$ randomly selects $H$ from the family of hash functions, and outputs $H$, together with either the hash value of $t$ or any distinct hash value, with the following probabilities:

$$\mathcal{R}_{\mathsf{OLH}}(t) = \begin{cases} \langle H, h \leftarrow H(t) \rangle, & \text{w/p } \frac{e^{\epsilon}}{e^{\epsilon}+g-1} \\ \langle H, h \overset{\$}{\leftarrow} [g] \backslash \{H(t)\} \rangle, & \text{w/p } \frac{g-1}{e^{\epsilon}+g-1} \end{cases}, \qquad (1)$$

where $\overset{\$}{\leftarrow}$ indicates uniform random sampling from the set $[g] \backslash \{H(t)\}$.

On receiving the perturbed value $y = \langle H, h \rangle$, $\bar{\mathbf{P}}_{\mathsf{OLH}}$ estimates its contribution to the frequency of $v \in \mathbb{D}$ as $\bar{\mathbf{P}}^{v}_{\mathsf{OLH}}(y) = \frac{1_{\{H(v)=h\}} - q}{p-q}$, where $p = \frac{e^{\epsilon}}{e^{\epsilon}+g-1}$ and $q = 1/g$. As stated in [31], answering the frequency for any $v \in \mathbb{D}$ using $\mathsf{OLH}$ has the error bound $\mathrm{MSE}(\bar{\mathbf{P}}^{v}_{\mathsf{OLH}}(\mathcal{R}(T))) = \mathrm{O}\left(\frac{4n}{\epsilon^2}\right)$.

$\mathsf{OLH}$ works well for point query, and its error for range query increases fast as the range volume increases. This motivates optimized multi-dimensional frequency oracles for range query.

### 3.1.2 Hierarchical-Interval Optimized Mechanism

The *hierarchical-interval optimized* ($\mathsf{HIO}$) mechanism [32] divides the domain $\mathbb{D}$ into hierarchy of nodes with various sizes on various layers. In particular, the hierarchy of a single ordinal attribute of cardinality $m$ consists of one interval of the entire attribute domain at the root and $m$ finest-grained intervals, *e.g.*, individual values, at the leaves, with intermediate intervals being the even split of their parents by the fan-out of $B$. For $d$ ordinal attributes, the $d$ hierarchies are cross-producted into one multi-dimensional hierarchy of nodes.

To perturb tuple $t$, $\mathcal{R}_{\mathsf{HIO}}$ samples one layer from the hierarchy and maps $t$ to the node that contain the tuple. The node is then perturbed by $\mathcal{R}_{\mathsf{OLH}}$ among nodes on its layer. To estimate the frequency of report $y$ for range predicate $\mathsf{C}$, $\bar{\mathbf{P}}^{\mathsf{C}}_{\mathsf{HIO}}$ first decomposes $\mathsf{C}$ into the set of hierarchical nodes $\mathcal{I}^{\mathsf{C}}$, and estimates the contribution of $y$ to each of the nodes using $\bar{\mathbf{P}}_{\mathsf{OLH}}$, if $y$ and the node are on the same layer. These frequencies, added together and multiplied by $\mathrm{O}(\log^d m)$, is the unbiased estimation of the frequency of $\mathsf{C}$. As stated in [32], answering the range frequency query $\mathsf{C}$ on $d_q$

attributes using HIO has the error bound:

$$\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{C}}_{\mathsf{HIO}}(\mathcal{R}(T))) = \mathrm{O}\left(\frac{n\log^{d+d_q} m}{\epsilon^2}\right). \quad (2)$$

[32] also achieves frequency oracles with nonsensitive weights for aggregation on what they call *measures* that are not sensitive.

## 3.2 Sensitive-weight Frequency Oracles

Consider a virtual fact table $T$ that is comprised of $n$ users' tuples. A query asks for the aggregation on a sensitive attribute $A$:

$$\texttt{SELECT}\ \mathsf{SUM}(A)\ \texttt{FROM}\ T\ \texttt{WHERE}\ \mathsf{C}. \quad (3)$$

This class of query requires the underlying LDP oracles to weight the frequency contribution of a perturbed value $y$ to $\mathsf{C}$ by its attribute value $A$, and we define the LDP primitive for such query $\mathsf{Q} = (A, \mathsf{C})$ as *sensitive-weight frequency oracle*, denoted as $\bar{\mathbf{P}}^{\mathsf{Q}}(y)$. Thus, query (3) can be estimated as $\bar{\mathbf{P}}^{\mathsf{Q}}(\mathcal{R}(T)) = \sum_{y\in\mathcal{R}(T)} \bar{\mathbf{P}}^{\mathsf{Q}}(y)$.

**Baseline I: HIO.** One baseline to sensitive-weight frequency oracles enumerates all possible values of $A$ and estimates their frequencies using multi-dimensional frequency oracles, *e.g.*, HIO, which are summed up, weighted by values of $A$, as the aggregation result. In particular, $\bar{\mathbf{P}}^{\mathsf{Q}}_{\mathsf{HIO}}(y) = \sum_{v\in A} v\cdot\bar{\mathbf{P}}^{\mathsf{C}\wedge(A=v)}_{\mathsf{HIO}}(y)$. According to the error bound of HIO (Equation (2)), the error bound of this baseline is: $\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{Q}}_{\mathsf{HIO}}(\mathcal{R}(T))) = \mathrm{O}\left(\sum_{v\in A} v^2\frac{n\log^{d+d_q} m}{\epsilon^2}\right)$.

**Baseline II: HIO with *stochastic rounding* (SR-HIO).** For improved utility, we can combine *stochastic rounding* (SR) [12] with HIO to enumerate only the two extreme domain values at aggregation. In particular, for each attribute $A$, $\mathcal{R}_{\mathsf{SR\text{-}HIO}}$ first rounds $t[A]$, *i.e.*, the value of $A$ in $t$, to either the min $A_{\min}$ or the max $A_{\max}$, using the *stochastic rounding* function $\mathcal{M}^A_{\mathsf{SR}}$ defined as follows:

$$\mathcal{M}^A_{\mathsf{SR}}(t[A]) = \begin{cases} A_{\min}, & \text{w/p } \frac{A_{\max}-t[A]}{A_{\max}-A_{\min}} \\ A_{\max}, & \text{w/p } \frac{t[A]-A_{\min}}{A_{\max}-A_{\min}} \end{cases}$$

Then, it perturbs the $d$ original attributes in $t$, together with the $d$ rounding values, using $\mathcal{R}_{\mathsf{HIO}}$ for $\epsilon$-LDP. Note that perturbing the $d$ original attribute values enables arbitrary range aggregation on the collected tuples while perturbing the $d$ rounding values enables the attribute aggregation on any of the $d$ attribute, both of which are necessary to support our target query class.

Given a perturbed value $y$, we can answer the frequency oracle for predicate $\mathsf{C}$ simply as $\bar{\mathbf{P}}^{\mathsf{C}}_{\mathsf{SR\text{-}HIO}}(y) = \bar{\mathbf{P}}^{\mathsf{C}}_{\mathsf{HIO}}(y)$, and, to answer query (3), we estimate the result by linearly combining the frequencies of the two extreme values, *i.e.*,

$$\bar{\mathbf{P}}^{\mathsf{Q}}_{\mathsf{SR\text{-}HIO}}(y) = A_{\min}\cdot\bar{\mathbf{P}}^{\mathsf{C}\wedge A_{\min}}_{\mathsf{HIO}}(y) + A_{\max}\cdot\bar{\mathbf{P}}^{\mathsf{C}\wedge A_{\max}}_{\mathsf{HIO}}(y).$$

The error bound of such an estimation for query (3) is

$$\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{Q}}_{\mathsf{SR\text{-}HIO}}(\mathcal{R}(T))) = \mathrm{O}\left(\frac{(A^2_{\min}+A^2_{\max})2^d n\log^{d+d_q} m}{\epsilon^2}\right),$$

because SR-HIO introduces $d$ extra attributes from rounding, each of which is of cardinality 2, and increases Equation (2) by multiplicative factor of $2^d$. Next, we propose the *partition-rounding-perturb* framework to eliminate the factor of $2^d$ in the error bound.

## 3.3 Partition-Rounding-Perturb Framework

The improvement of SR-HIO comes from the fact that only one attribute can appear in the aggregation function F. To leverage such fact, we first randomly *partition* the tuples into $d$ groups, one for each of the $d$ attributes, using a randomized partition function $\mathcal{S}$, *i.e.*, $\mathcal{S}(t) = t[A]$, for $A \overset{\$}{\leftarrow} \{A_1,\cdots,A_d\}$, where $\overset{\$}{\leftarrow}$ indicates uniform random sampling from the $d$ attributes; then apply *rounding* function $\mathcal{M}^A_{\mathsf{SR}}$ on the attribute from $\mathcal{S}$ for each tuple independently; finally, for each tuple, we *perturb* its $d$ attribute values, together with the single rounding value. Now to aggregate on attribute $A$, we only use tuples partitioned for $A$, aggregate their estimated contributions, and scale the result by $d$ as the final result.

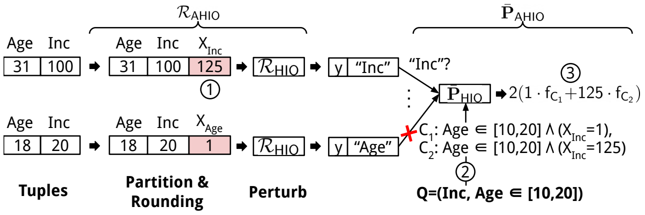

$\mathcal{R}_{\mathsf{AHIO}}$
$\bar{\mathbf{P}}_{\mathsf{AHIO}}$
Age Inc
31 100
Age Inc X_Inc
31 100 125
①
$\mathcal{R}_{\mathsf{HIO}}$
y "Inc"
"Inc"?
③
$\bar{\mathbf{P}}_{\mathsf{HIO}}$
$2(1\cdot f_{C_1}+125\cdot f_{C_2})$
Age Inc
18 20
Age Inc X_Age
18 20 1
$\mathcal{R}_{\mathsf{HIO}}$
y "Age"
C1: Age ∈ [10,20] ∧ (X_Inc=1),
C2: Age ∈ [10,20] ∧ (X_Inc=125)
②
Tuples
Partition & Rounding
Perturb
Q=(Inc, Age ∈ [10,20])

Figure 4: Example of running AHIO on tuples with attributes Age $\in [1,125]$ and Inc $\in [1,125]$ (income), and aggregation on Inc for users with Age $\in [10,20]$. ① partition the tuples by attribute, round their Inc or Age value, and augment with attribute $X_{\mathsf{Inc}}$ or $X_{\mathsf{Age}}$ for the rounding value; ② rewrite the query into two frequency aggregations for the min and max of Inc; ③ combine the frequency estimations, weighted by the min and max of Inc, and scale the result by 2 to compensate for the attribute partitioning.

We call the overall framework *partition-rounding-perturb* (PRP), and, next, we introduce two instantiations using different ways to perturb the extra rounding value, with different error bounds.

### 3.3.1 *Augment-then-Perturb* (AHIO)

AHIO augments the tuple $t$ with the extra attribute $X_A$ for the rounding value of attribute $A$, and $\mathcal{R}_{\mathsf{AHIO}}$ perturbs $t$, including the value of $X_A$, using HIO. Overall, we have:

$$\mathcal{R}_{\mathsf{AHIO}}(t) = \mathcal{R}_{\mathsf{HIO}}(\langle t, X_A = \mathcal{M}^A_{\mathsf{SR}}(\mathcal{S}(t))\rangle).$$

At aggregation, $\bar{\mathbf{P}}_{\mathsf{AHIO}}$ estimates the frequencies of tuples that satisfy $\mathsf{C}$ and have certain rounding value by patching $\mathsf{C}$ with conjunctive equality condition for the rounding value. For instance, to estimate the frequency of tuples that satisfy $\mathsf{C}$ and have attribute $A$ rounded to $A_{\min}$, $\bar{\mathbf{P}}_{\mathsf{AHIO}}$ answers the multi-dimensional frequency using $\bar{\mathbf{P}}_{\mathsf{HIO}}$ on condition $\mathsf{C} \wedge X_A = A_{\min}$. Then it adds the frequencies, multiplied by the corresponding rounding values and $d$, to answer the sensitive-weight frequency oracles, *i.e.*, :

$$\bar{\mathbf{P}}^{\mathsf{Q}}_{\mathsf{AHIO}}(y) = \begin{cases} 0, & \text{if } y \text{ is not in the group for } A \\ d(A_{\min}\cdot\bar{\mathbf{P}}^{\mathsf{C}\wedge(X_A=A_{\min})}_{\mathsf{HIO}}(y) & \\ +A_{\max}\cdot\bar{\mathbf{P}}^{\mathsf{C}\wedge(X_A=A_{\max})}_{\mathsf{HIO}}(y)), & \text{otherwise} \end{cases}$$

Figure 4 shows how AHIO works using a toy example with two attributes and an aggregation query. The error bound of AHIO is:

**Lemma 4.** Answering query (3) using AHIO under $\epsilon$-LDP has the error bound of:

$$\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{Q}}_{\mathsf{AHIO}}(\mathcal{R}(T))) = \mathrm{O}\left(\frac{2(A^2_{\min}+A^2_{\max})n\cdot d\log^{d+d_q} m}{\epsilon^2}\right).$$

**Proof Sketch:** First, since the $n$ tuples is partitioned into $d$ groups, and the augmented attribute introduces an hierarchy of 2 layers, using HIO with $\epsilon$-LDP to estimate the frequencies for $\mathsf{C} \wedge (X_A = A_{\min})$ and $\mathsf{C}\wedge(X_A = A_{\max})$ has error bound of $\mathrm{O}\left(\frac{2n\log^{d+d_q} m}{d\epsilon^2}\right)$. Then, scaling the frequencies by $A_{\min}$ and $A_{\max}$, and multiplying the weighted frequency sum by $d$, introduces multiplicative factors $(A^2_{\min} + A^2_{\max})$ and $d^2$ to the error. □

### 3.3.2 *Embed-then-Perturb* (EHIO)

EHIO embeds the rounding value into the partition attribute by doubling its domain. Specifically, it doubles the domain of $A$ to $A^-(= [2A_{\min} - A_{\max} - 1, A_{\min} - 1]) \vee A^+(= [A_{\min}, A_{\max}])$. If the rounding value is $A_{\min}$, $\mathcal{R}_{\mathsf{EHIO}}$ maps $t[A]$ to $2A_{\min} - t[A] - 1 \in A^-$; otherwise, the value is unchanged, and in $A^+$. And the mapped tuple is perturbed by $\mathcal{R}_{\mathsf{HIO}}$.

To aggregate on $A$, $\bar{\mathbf{P}}_{\mathsf{EHIO}}$ patches range conditions, *i.e.*, $A \in A^-$ for $A_{\min}$ and $A \in A^+$ for $A_{\max}$, to the predicate $\mathsf{C}$ to estimate the frequencies of the two rounding values:

$$\bar{\mathbf{P}}^{\mathsf{Q}}_{\mathsf{EHIO}}(y) = \begin{cases} 0, & \text{if y is not partitioned for A} \\ d(A_{\min}\cdot\bar{\mathbf{P}}^{\mathsf{C}\wedge(A\in A^-)}_{\mathsf{HIO}}(y) & \\ +A_{\max}\cdot\bar{\mathbf{P}}^{\mathsf{C}\wedge(A\in A^+)}_{\mathsf{HIO}}(y)), & \text{otherwise} \end{cases}$$

If C contains range condition $A \in [l, r]$, EHIO maps it to conditions $A \in [2A_{\min} - l - 1, 2A_{\min} - r - 1]$ and $A \in [l, r]$ for $A_{\min}$ and $A_{\max}$, respectively. EHIO doubles the domain size of attribute $A$ from $m$ to $2m$. Thus, its error bound for query (3) is:

$$\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{Q}}_{\mathsf{EHIO}}(\mathcal{R}(T))) = \mathrm{O}\left(\frac{(A_{\min}^2 + A_{\max}^2) n d \log^2 2m \log^{d+d_q-2} m}{\epsilon^2}\right).$$

**Remarks.** The difference between the error bounds of AHIO and EHIO is $\frac{2\log^2 m}{\log^2 2m}$, which is between $0.5$ (when $m = 2$) and $2$ (when $m \to \infty$). It is better to use AHIO and EHIO when $m$ is small and large, respectively.

# 4. 1-TO-1 JOINT FREQUENCY ORACLES

In this section, we focus on the joint aggregation over tuples that are collected by $K$ separate services:

$$\texttt{SELECT}\ \mathsf{F}(A)\ \texttt{FROM}\ T_1\ \texttt{JOIN}\ \ldots\ \texttt{JOIN}\ T_K\ \texttt{WHERE}\ \mathsf{C}, \qquad (4)$$

where $A$ is the aggregation attribute collected by one of the joining services, and C consists of $K \cdot d_q$ conditions, on $d_q$ attributes of each service. The key task here is to, given the vector of perturbed values $\mathbf{y} = \langle y_i \rangle_{i=1}^K$ joined on the same user id, estimate its contribution to the frequency of C, and we call such LDP primitive *joint frequency oracles*, denoted as $\bar{\mathbf{P}}^{\mathsf{C}}(\mathbf{y})$. Given such frequency oracles, we can plug it into the partition-rounding-perturb framework, in place of HIO, to achieve attribute aggregation, *i.e.*, $\bar{\mathbf{P}}^{\mathsf{Q}}(\mathbf{y})$, for $\mathsf{Q} = (A, \mathsf{C})$, *i.e.*, $\bar{\mathbf{P}}^{\mathsf{Q}}(\bowtie_{i=1}^K \mathcal{R}(T_i)) = \sum_{\mathbf{y} \in \bowtie_{j=i}^K \mathcal{R}(T_i)} \bar{\mathbf{P}}^{\mathsf{Q}}(\mathbf{y})$.

Next, we focus on joint frequency oracles over tuples with one-to-one relation, and extend to one-to-many relation in Section 5.

## 4.1 Split-and-Conjunction Baseline

Split-and-conjunction (SC) in [32] enables single-table multi-dimensional frequency oracles over tuples with independently perturbed attributes. In our setting, we can use SC to independently perturb all attributes from the $K$ services, each with privacy parameter $\frac{\epsilon}{K \cdot d}$, and estimate the frequency of C using the reported perturbed attribute values. We call such mechanism SC-JOIN.

To estimate the frequency for C, SC-JOIN first decomposes C into $K \cdot d_q$ atomic predicates $\mathsf{C}_1, \mathsf{C}_2, \ldots$, one for each attribute in C. For the joined value $\mathbf{y}$, SC-JOIN increments $\mathbf{f}'_{\mathsf{C}}[x]$ by one, where the $i$-th bit of $x$ is 1 if $\mathbf{y}$ satisfies $\mathsf{C}_i$; or 0 otherwise. Similarly, we can denote the vector $\mathbf{f}_{\mathsf{C}}$ as the vector of frequencies on the original tuples, and $\mathbf{f}_{\mathsf{C}}[11 \ldots 1]$, *i.e.*, 1 for all $\mathsf{C}_i$ in C, is the true frequency aggregation for C. The two vectors are connected via a state transition matrix $\mathbf{M}$, which defines the stochastic transition from the state $\mathbf{f}_{\mathsf{C}}$ to the frequency distribution $\mathbf{f}'_{\mathsf{C}}$ *i.e.*, $\mathbf{E}[\mathbf{f}'_{\mathsf{C}}] = \mathbf{M} \cdot \mathbf{f}_{\mathsf{C}}$. For instance, when C contains only one condition, we have

$$\mathbf{E}\left[\begin{pmatrix}\mathbf{f}'_{\mathsf{C}}[0] \\ \mathbf{f}'_{\mathsf{C}}[1]\end{pmatrix}\right] = \begin{pmatrix}\Pr[0|0] & \Pr[0|1] \\ \Pr[1|0] & \Pr[1|1]\end{pmatrix}\begin{pmatrix}\mathbf{f}_{\mathsf{C}}[0] \\ \mathbf{f}_{\mathsf{C}}[1]\end{pmatrix},$$

where $\Pr[x'|x]$ indicates the probabilities of the perturbed value evaluated as true/false ($x' = 1/0$), given the original value evaluated as true/false ($x = 1/0$). Generally, for C with conjunctive conditions on $K \cdot d_q$ attributes, $\mathbf{M}$ is of dimension $2^{K \cdot d_q} \times 2^{K \cdot d_q}$, and we can derive $\mathbf{M}$ based on the probabilities $p$ and $q$ of the LDP mechanism (See [32] for details). With $\mathbf{f}'_{\mathsf{C}}$ and $\mathbf{M}$, we can calculate $\mathbf{M}^{-1} \cdot \mathbf{f}'_{\mathsf{C}}$ as the unbiased estimation of the initial state $\mathbf{f}_{\mathsf{C}}$, which contains the frequency for the given predicate. We call such technique *state inversion*, and we have $\bar{\mathbf{P}}^{\mathsf{C}}_{\mathsf{SC\text{-}JOIN}}(\mathbf{y}) = (\mathbf{M}^{-1} \cdot \mathbf{f}'_{\mathbf{y},\mathsf{C}})[11 \ldots 1]$, where $\mathbf{f}'_{\mathbf{y},\mathsf{C}}$ is the vector $\mathbf{f}'_{\mathsf{C}}$ on $\mathbf{y}$.

As stated in [32], for $\mathsf{Q} = (A, \mathsf{C})$ on $T = \bowtie_{i=1}^K T_i$, SC-JOIN has error bound of: $\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{C}}_{\mathsf{SC\text{-}JOIN}}(\mathcal{R}(T))) = \mathrm{O}\left(\frac{n \log^{3K \cdot d_q} m}{(\epsilon/(K \cdot d))^{2K \cdot d_q}}\right)$.

To answer the joint frequency of C, SC-JOIN requires joint estimation on all attributes in C, whether from the same or different services. For better utility, we want to minimize the number of joint estimations to the number of services involved in C, *i.e.*, up to $K$.

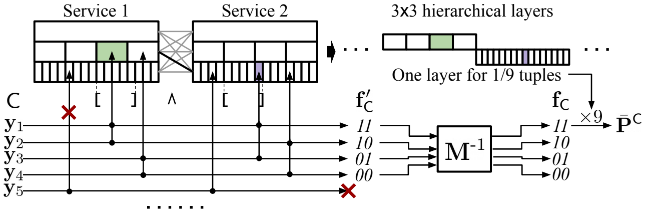


Figure 5: HIO-JOIN across two services, each collecting one attribute using hierarchy with fan-out $B = 4$.

## 4.2 Multi-Service Joint Frequency Oracles

While supporting joint analysis, SC-JOIN independently perturbs all the attributes. This makes single-table analysis more noisy as SC is used for each service, instead of HIO. To improve the utility for single-service analysis and at the same time support joint analysis, we propose to adapt the state inversion technique in SC-JOIN for HIO reports, and call such mechanism HIO-JOIN.

For each tuple of a service, $\mathcal{R}_{\mathsf{HIO\text{-}JOIN}}$ applies $\mathcal{R}_{\mathsf{HIO}}$ to perturb it. Formally, for user $u$ with tuples $T_1^u, \ldots, T_K^u$ across the $K$ services,

$$\mathcal{R}_{\mathsf{HIO\text{-}JOIN}}(T_1^u, \ldots, T_K^u) = \mathcal{R}^{\epsilon/K}_{\mathsf{HIO}}(T_1^u), \ldots, \mathcal{R}^{\epsilon/K}_{\mathsf{HIO}}(T_K^u) \qquad (5)$$

is reported, where $\mathcal{R}^{\epsilon/K}_{\mathsf{HIO}}(T_i^u)$ perturbs the single tuple in $T_i^u$ using HIO with privacy parameter $\frac{\epsilon}{K}$. Here, tuples of the $K$ services are independently perturbed by $\mathcal{R}_{\mathsf{HIO}}$, and the privacy parameter $\epsilon$ is evenly divided for all the $K$ tuples of the user so that the overall privacy loss is bounded by $\epsilon$. Next, we focus on the joint aggregation over these perturbed HIO reports.

We adapt the state inversion technique in SC-JOIN for HIO-JOIN with the key insight that, after mapping a tuple to the node in the hierarchy, $\mathcal{R}_{\mathsf{HIO}}$ perturbs the node in the exact same manner as SC perturbs the single attribute. Thus, for each hierarchical layer across the $K$ services, we can derive the state transition matrix for the nodes on that layer across the $K$ services, following what SC does for nodes across the $K \cdot d_q$ attributes, which gives us a state transition matrix of dimension $2^K \times 2^K$. The joined tuples that are sampled by $\mathcal{R}_{\mathsf{HIO}}$ on the same layers as the predicate decompositions are classified into the vector $\mathbf{f}'_{\mathsf{C}}$, which is then multiplied by $\mathbf{M}^{-1}$ to recover the frequency of the original tuples. The estimated frequencies of all the decompositions are added together, and scaled by the number of layers across $K$ services, *i.e.*, $\log^{K \cdot d} m$, to compensate for the layer sampling in HIO. Thus, we have the frequency estimation of HIO-JOIN as:

$$\bar{\mathbf{P}}^{\mathsf{C}}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y}) = \begin{cases} 0, & \text{if } \mathbf{y} \text{ and C on different layers} \\ \log^{K \cdot d} m \cdot (\mathbf{M}^{-1} \cdot \mathbf{f}'_{\mathbf{y},\mathsf{C}})[11 \ldots 1], & \text{otherwise} \end{cases}$$

Figure 5 shows how HIO-JOIN works with two services, each collecting one attribute: at collection, each tuple of a service is perturbed on one the 3 layers of the hierarchy of the attribute; at aggregation, the predicate C with range conditions on attributes of the two services is first decomposed into pairs of nodes across the two hierarchies. For one pair of nodes, *i.e.*, the green and purple ones, HIO-JOIN calculates $\mathbf{f}'_{\mathsf{C}}$ by counting the numbers for indices 11, 10, 01 and 00, which represent the four cases where the joined tuple $\mathbf{y}_i$ has part from service 1 (not) in the green node and part from service 2 (not) in the purple node. For instance, $\mathbf{y}_1$ has its two attributes inside the green and purple nodes, which increments $\mathbf{f}'_{\mathsf{C}}[11]$ by 1. And $\mathbf{y}_2$ has its first attribute inside the green node, and the second attribute outside the purple node, which increments $\mathbf{f}'_{\mathsf{C}}[10]$ by 1. Joined values not on the same layers as the decomposed nodes, *e.g.*, $\mathbf{y}_5$, do not contribute to any of the classes. Since there are $3 \times 3$ layers across the two hierarchies, $\frac{1}{9}$ of the collected tuples are expected to be on the same layer as the decomposed nodes. Thus, after multiplying $\mathbf{f}'_{\mathsf{C}}$ by $\mathbf{M}^{-1}$, HIO-JOIN multiplies the estimation by 9. HIO-JOIN repeats the estimation for the other decompositions of predicate C, and adds their estimations together as the aggregation for C.

We state the error bound of HIO-JOIN in Lemma 5:

**Lemma 5.** Answering the frequency query for predicate C on $K \cdot d_q$ attributes against $T = \bowtie_{i=1}^{K} T_i$ using HIO-JOIN has error bound:

$$\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{C}}_{\textsf{HIO-JOIN}}(\mathcal{R}(T))) = \mathrm{O}\Big(\frac{n \log^{K(d+d_q)} m}{(\epsilon/K)^{2K}}\Big).$$

**Proof Sketch:** First, the number of joined values sampled on the same layer as C is $\mathrm{O}\Big(\frac{n}{\log^{K \cdot d} m}\Big)$. For these joined values, since the parameter $\epsilon$ is only splitted by $K$, estimating the range frequency on each of the $\mathrm{O}\big(\log^{K \cdot d_q} m\big)$ hierarchy nodes, using model inversion, incurs error bound of $\mathrm{O}\Big(\frac{n}{\log^{K \cdot d} m(\epsilon/K)^{2K}}\Big)$. Finally, we multiply the aggregations by factor of $\mathrm{O}\big(\log^{K \cdot d} m\big)$. □

Note that here the effect of joint estimation, which is manifested as the power of $\epsilon$, is at the scale of $K$, which is smaller than the $K \cdot d_q$ in SC-JOIN. The difference will be considerable when $d_q$ is large and $\epsilon$ is small, which we evaluate in Section 7.3

**Security.** HIO-JOIN splits $\epsilon$ by $K$, and, by sequential composition, the overall privacy guarantees for each user is $\epsilon$-LDP. Furthermore, since users have the same number of tuples collected by each service, *i.e.*, 1, it is $\epsilon$-uLDP. This argument, however, does not generalize to the one-to-many relation, where users of the $i$-th service can generate arbitrary numbers of tuples. We will address this problem in Section 5.

# 5. HANDLING ONE-TO-MANY JOIN

In this section, we focus on the joint aggregation over tuples with one-to-many relation. In particular, we focus on the primary-foreign-key joint aggregation with two services. Each user $u$ generates exactly one tuple with user id as the primary key in service 1, *i.e.*, $\forall u : |T_1^u| = 1$, and up to $\tau_{\max}$ tuples with user id as a foreign key in service 2, *i.e.*, $\forall u : 0 \leqslant |T_2^u| \leqslant \tau_{\max}$. We assume that $\tau_{\max}$, the maximum number of tuples that can be generated by a user in service 2 is public. We want to guarantee $\epsilon$-uLDP (Definition 3), and the target analytical task is the same as query (4).

## 5.1 Frequency-based Attack

We introduced SC-JOIN and HIO-JOIN for joint aggregation under one-to-one relation. A straightforward extension to one-to-many relation is to split the privacy parameter $\epsilon$ for a user $u$ among all the tuples collected. That is, each tuple in services 1 and 2 is reported under $\epsilon/(|T_1^u| + |T_2^u|)$-LDP. And the overall privacy guarantees for $u$ is $\epsilon$-LDP, based on the sequential composition.

Unfortunately, this approach is not private, as it enables the simple yet effective *frequency-based attack*: After collecting perturbed tuples from service 2, for each user, the adversary can count the number (*frequency*) of tuples that can be joined with some tuple from service 1 on user id, by grouping the tuples by user id. With such frequency information available, one can immediately distinguish between users with different usage patterns in service 2. With extra prior knowledge, based on these frequencies, the adversary can launch even more devastating attacks to infer sensitive attributes about users (see Example 1.1). In fact, the presence of some reported tuples or absence of any tuple already reveals information about whether the user is using a service, which by itself is sensitive especially when the service targets certain group of users.

More formally, we can show that the above approach does not provide uLDP (Definition 3) to individual users. To show this for HIO-JOIN and two users $u$ and $u'$, s.t., $|T_2^u| \neq |T_2^{u'}|$, we take $\mathbf{y} = \mathcal{R}_{\textsf{HIO-JOIN}}(T^u)$ in Definition 3, and, thus, $|\mathbf{y}| = |\mathcal{R}_{\textsf{HIO-JOIN}}(T^u)| \neq |\mathcal{R}_{\textsf{HIO-JOIN}}(T^{u'})|$. Thus, the ratio between the probabilities of the perturbed tuples from $u$ and $u'$ being equal to $\mathbf{y}$ are unbounded ($\infty$-uLDP), as $\mathbf{Pr}[\mathcal{R}(T^u) = \mathbf{y}] > 0$, and $\mathbf{Pr}[\mathcal{R}(T^{u'}) = \mathbf{y}] = 0$.

The problem of the above approach is that it outputs a perturbed value for each input tuple of the user, and breaks the privacy guarantees when users have different numbers of tuples. Next, we close such security loophole by hiding the real number of tuples with user id as the foreign-key, *i.e.*, from service 2, of a user, and achieve $\epsilon$-uLDP under one-to-many relation.

## 5.2 Hiding Existence and Frequency with $\tau$-Truncation

To prevent the frequency-based attacks that exploit the distinct numbers of collected tuples on service 2 among users, we propose the $\tau$-truncation mechanism that outputs the fixed number, *i.e.*, $\tau$, of perturbed tuples for each user's tuples on service 2.

Suppose each user $u$ generates a maximum of $\tau_{\max}$ tuples in service 2, *i.e.*, $0 \leqslant |T_2^u| \leqslant \tau_{\max}$. Here, $|T_2^u| = 0$ means that $u$ is not using this service. The goal of $\tau$-truncation is to hide each user's presence/absence information as well as the true frequency.

If $|T_2^u| > 0$, we want to hide the frequency $|T_2^u|$. The mechanism samples $\tau$ tuples from $T_2^u$ with replacement, and attach a *truncation weight* (inversed sampling ratio) $r = |T_2^u|/\tau \in [r_{\min}, r_{\max}]$, where $r_{\min} = 0$ and $r_{\max} = \tau_{\max}/\tau$, to each of these tuples. This weight is used to obtain an unbiased estimate of the aggregation.

If $|T_2^u| = 0$, we want to hide the absence of $u$. The mechanism randomly draws a sample of $\tau$ tuples from the domain $\mathbb{D}_2$ of $T_2$, and attach a weight $r = 0$ to each of them (so that we know these dummy tuples have no contribution in any aggregation).

Formally, the $\tau$-truncation procedure Trunc is defined to be:

$$\mathsf{Trunc}(\tau, T) = \begin{cases} \langle t_i \xleftarrow{\$} T, r = \frac{|T|}{\tau} \rangle_{i=1}^{\tau} & \text{if } |T| > 0 \\ \langle t_i \xleftarrow{\$} \mathbb{D}, r = 0 \rangle_{i=1}^{\tau} & \text{if } |T| = 0 \end{cases},$$

where $t_i \xleftarrow{\$} T$ (or the domain $\mathbb{D}$ of $T$) means that we use uniform random sampling to draw a tuple from $T$ (or $\mathbb{D}$) as $t_i$.

In both cases, each user generates exactly $\tau$ tuples $\langle t_i, r \rangle_{i=1}^{\tau}$. In order to completely hide the existence and frequency information as well as the content in $t_i$, we apply an LDP perturbation mechanism, *e.g.*, $\mathcal{R}_{\textsf{AHIO}}$ in Section 3.3, on both the value of $r$ and $t_i$.

In terms of service 1, since user id is the primary key, each user generates exactly one tuple in $T_1^u$, and thus, we only need to apply the perturbation mechanism on this tuple.

For each user $u$, there are a total of $\tau + 1$ tuples to be perturbed, namely, one tuple in $T_1^u$ and $\tau$ tuples in $\mathsf{Trunc}(\tau, T_2^u) = \langle t_i, r \rangle_{i=1}^{\tau}$. The privacy budget is evenly partitioned among these $\tau + 1$ tuples. Putting them together, we are going to release

$$\mathcal{R}^{\epsilon}_{\textsf{Trunc}}(T_1^u, T_2^u) = \mathcal{R}^{\epsilon/(\tau+1)}_{\textsf{AHIO}}(T_1^u) \oplus \langle \mathcal{R}^{\epsilon/(\tau+1)}_{\textsf{AHIO}}(\langle t_i, r \rangle) \rangle_{i=1}^{\tau} \quad (6)$$

where each instance of $\mathcal{R}^{\epsilon/(\tau+1)}_{\textsf{AHIO}}$ with privacy parameter $\frac{\epsilon}{\tau+1}$ runs independently. We can show that $\mathcal{R}^{\epsilon}_{\textsf{Trunc}}$ is $\epsilon$-uLDP.

**Lemma 6.** Collecting a user's tuples from service 1 (with user id as the primary key) and service 2 (with user id as a foreign key) using $\mathcal{R}^{\epsilon}_{\textsf{Trunc}}$ above satisfies $\epsilon$-uLDP.

**Proof Sketch:** For users $u$ and $u'$, they both have one tuple on service 1, and $|T_2^u|$ and $|T_2^{u'}|$ tuples on service 2, respectively. With $\tau$-truncation, both users have $\tau$ tuples collected by service 2. In addition, all collected tuples of a user is perturbed with $\frac{\epsilon}{1+\tau}$-LDP. In the definition of uLDP, any possible value $\mathbf{y}$ from $\tau$-truncation consists of one perturbed tuple for service 1 and $\tau$ perturbed tuples for service 2. Thus, we have $\frac{\mathbf{Pr}[\mathcal{R}(T^u)=\mathbf{y}]}{\mathbf{Pr}[\mathcal{R}(T^{u'})=\mathbf{y}]} \leqslant (e^{\epsilon/(1+\tau)})^{1+\tau}$. □

Both the aggregating attribute $A$ and the truncation weight $r$ are randomly rounded (to $\{A_{\min}, A_{\max}\}$ and $\{r_{\min}, r_{\max}\}$, respectively) during the perturbation process $\mathcal{R}_{\textsf{AHIO}}$. In the next subsection, we will introduce our *double-rounding mechanism and estimation* technique to recover the answer to the original aggregation query from truncated and perturbed data in an unbiased way.

## 5.3 Double Rounding: Recovering Aggregation from Truncated Tuples

At aggregation, the contribution of each perturbed value need to be multiplied by its truncation weight to compensate for tuples truncated away by Trunc. In particular, we need to answer the following queries for $\mathsf{COUNT}(*)$ and $\mathsf{SUM}(A)$, respectively:

$$\texttt{SELECT}\ \mathsf{SUM}(r)\ \texttt{FROM}\ T_1\ \texttt{JOIN}\ T_2\ \texttt{WHERE}\ \mathsf{C},\ \text{and} \tag{7}$$

$$\texttt{SELECT}\ \mathsf{SUM}(r \cdot A)\ \texttt{FROM}\ T_1\ \texttt{JOIN}\ T_2\ \texttt{WHERE}\ \mathsf{C}, \tag{8}$$

where $r$ is the truncation weight in the collected tuples. Queries (7) and (8) require two sensitive-weight frequency oracles, *i.e.*, one for $\mathsf{Q} = (r, \mathsf{C})$ and the other for $\mathsf{Q} = (r \cdot A, \mathsf{C})$, and directly applying the mechanisms in Section 3.3 would degrade the utility for the frequency aggregation (query (7)) by $\mathrm{O}(d + 1)$, due to partitioning.

To preserve the utility for frequency aggregation, we do not partition the tuples for frequency aggregation, and only partition the tuples among the $d$ attributes for attribute aggregation, via what we call *double rounding*: for each truncated tuple output by Trunc, we group it for one of the the $d$ attributes, and derive two rounding values, one for the partition attribute $A$, and the other for the truncation weight $r$; thus, the rounding value for $r$ enables query (7) for all tuples, *i.e.*, $\bar{\mathbf{P}}^{(r,\mathsf{C})}_{\mathsf{Trunc}}(\mathbf{y}) = r_{\max} \cdot \bar{\mathbf{P}}^{\mathsf{C} \wedge r_{\max}}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y}) + r_{\min} \cdot \bar{\mathbf{P}}^{\mathsf{C} \wedge r_{\min}}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y})$; Note that, since $r_{\min} = 0$, the contributions for predicate $\mathsf{C} \wedge r_{\min}$ is zero, and, thus, ignored; similarly, the rounding values for $A$ and $r$ together enable query (8) for tuples partitioned for $A$, and, for one such $\mathbf{y}$, we have $\bar{\mathbf{P}}^{(r \cdot A,\mathsf{C})}_{\mathsf{Trunc}}(\mathbf{y}) = d \cdot \sum_{b_r} \sum_{b_A} b_r \cdot b_A \cdot \bar{\mathbf{P}}^{\mathsf{C} \wedge b_r \wedge b_A}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y})$, where $b_r \in \{r_{\min}, r_{\max}\}$ and $b_A \in \{A_{\min}, A_{\max}\}$.

Concretely, we pack the two rounding values as a pair, and augment the tuple with attribute $X_{r,A} \in \{\langle r_{\min}, A_{\min}\rangle, \langle r_{\min}, A_{\max}\rangle, \langle r_{\max}, A_{\min}\rangle, \langle r_{\max}, A_{\max}\rangle\}$ for it. The augmented attribute is perturbed, together with the original $d$ attributes, for data collection. For $\bar{\mathbf{P}}^{(r \cdot A,\mathsf{C})}_{\mathsf{Trunc}}(\mathbf{y})$, if $\mathbf{y}$ is partitioned for $A$, we estimate

$$\bar{\mathbf{P}}^{\mathsf{C} \wedge r_{\min} \wedge A_{\min}}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y}) = \bar{\mathbf{P}}^{\mathsf{C} \wedge (X_{r,A} = \langle r_{\min}, A_{\min}\rangle)}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y}),$$

as well as for other pairs: $\langle r_{\min}, A_{\max}\rangle$, $\langle r_{\max}, A_{\min}\rangle$ and $\langle r_{\max}, A_{\max}\rangle$; otherwise, $\bar{\mathbf{P}}^{(r \cdot A,\mathsf{C})}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y}) = 0$. And we have:

**Lemma 7.** $\bar{\mathbf{P}}^{(r \cdot A,\mathsf{C})}_{\mathsf{Trunc}}(\mathcal{R}_{\mathsf{Trunc}}(T_1, T_2))$ is unbiased for query (8).

**Proof Sketch:** First, since $\bar{\mathbf{P}}_{\mathsf{HIO\text{-}JOIN}}$ is unbiased, and $r$ and $A$ are independently rounded, we have

$$\mathbf{E}[\bar{\mathbf{P}}^{\mathsf{C} \wedge (X_{r,A} = \langle r_{\min}, A_{\min}\rangle)}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y})] = \tfrac{1_{\{\mathbf{t} \in \mathsf{C}\}} \cdot (r_{\max} - \mathbf{t}[r])(A_{\max} - \mathbf{t}[A])}{d(r_{\max} - r_{\min})(A_{\max} - A_{\min})}.$$

And we have similar results for $\langle r_{\min}, A_{\max}\rangle$, $\langle r_{\max}, A_{\min}\rangle$ and $\langle r_{\max}, A_{\max}\rangle$. Hence, we have $\mathbf{E}[\bar{\mathbf{P}}^{(r \cdot A,\mathsf{C})}_{\mathsf{Trunc}}(\mathcal{R}_{\mathsf{Trunc}}(\langle T_1, T_2\rangle))]$

$$= \textstyle\sum_{\mathbf{t} \in T_1 \bowtie T_2} \tfrac{d \cdot 1_{\{\mathbf{t} \in \mathsf{C}\}} \cdot \mathbf{t}[r] \cdot \mathbf{t}[A] \cdot (r_{\max} - r_{\min})(A_{\max} - A_{\min})}{d \cdot (r_{\max} - r_{\min})(A_{\max} - A_{\min})}$$

$$= \textstyle\sum_{\mathbf{t} \in T_1 \bowtie T_2} 1_{\{\mathbf{t} \in \mathsf{C}\}} \cdot \mathbf{t}[r] \cdot \mathbf{t}[A]. \qquad \square$$

**Lemma 8.** For $n$ users, whose tuples on service 1 and 2 are collected via $\mathcal{R}_{\mathsf{Trunc}}$, answering query (8) using $\bar{\mathbf{P}}_{\mathsf{Trunc}}$ has error bound:

$$\mathrm{O}\left(\tfrac{nd\tau(1+\tau)^4 \log^{2(d+d_q)} m}{\epsilon^4}(r_{\min}^2 + r_{\max}^2)(A_{\min}^2 + A_{\max}^2)\right). \tag{9}$$

**Proof Sketch:** First, since the $\epsilon$ is splitted evenly for the $1 + \tau$ tuples of a user, the tuples are each perturbed with $\frac{\epsilon}{1+\tau}$-LDP. In addition, we have $r_{\min} = 0, r_{\max} = \tau_{\max}/\tau$. Thus, estimating the contribution of one pair of joined tuples from the two services to the aggregation on $A$, using $\bar{\mathbf{P}}^{(r \cdot A,\mathsf{C})}_{\mathsf{HIO\text{-}JOIN}}(\mathbf{y})$, has error bound

$$\mathrm{O}\left(\tfrac{d(1+\tau)^4 \log^{2(d+d_q)} m}{\epsilon^4}(r_{\min}^2 + r_{\max}^2)(A_{\max}^2 + A_{\min}^2)\right).$$

We need to add up the contributions from $n \cdot \tau$ pairs of joined tuples, which sums up the error as Equation (9). $\square$

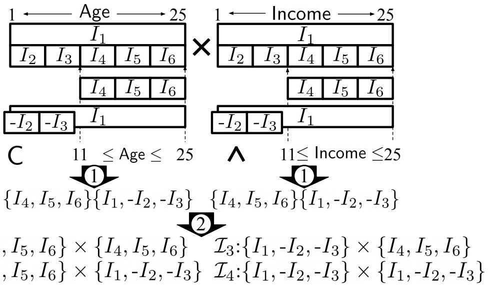

1
Age
25
1
Income
25
$I_1$
$I_2$ $I_3$ $I_4$ $I_5$ $I_6$
×
$I_4$ $I_5$ $I_6$
-$I_2$ -$I_3$
C
11 ≤ Age ≤ 25
∧
11≤ Income ≤25
1
2
$\{I_4, I_5, I_6\}\{I_1, \text{-}I_2, \text{-}I_3\}$
$\mathcal{I}_1$:$\{I_4, I_5, I_6\} \times \{I_4, I_5, I_6\}$
$\mathcal{I}_3$:$\{I_1, \text{-}I_2, \text{-}I_3\} \times \{I_4, I_5, I_6\}$
$\mathcal{I}_2$:$\{I_4, I_5, I_6\} \times \{I_1, \text{-}I_2, \text{-}I_3\}$
$\mathcal{I}_4$:$\{I_1, \text{-}I_2, \text{-}I_3\} \times \{I_1, \text{-}I_2, \text{-}I_3\}$

Figure 6: Optimal range decompositions for predicate C on attributes Age and Income, with fan-out of 5.

Based on Lemma 8, setting $\tau = 1$ achieves the optimal error bound. We evaluate the effects of $\tau$ on utility in Section 7.3.

# 6. RANGE CONSISTENCY OPTIMIZATION

For single dimensional range aggregation, [8, 22] show that post-processing the hierarchy for *consistency*, *i.e.*, equality between aggregation on every node and the aggregations on its children, improves the aggregation utility. For multi-dimensional case, however, it is challenging to achieve the consistency without exponentially (to the number of attributes) increasing the post-processing overhead. In this section, we propose a cost-effective consistency optimization technique based on optimal range decompositions.

## 6.1 Optimal Range Decomposition

Given the multi-dimensional hierarchy, there are multiple ways to decompose the predicate C into the (hierarchy) nodes. For instance, in Figure 6, the predicate asks for $\mathsf{C} = \mathsf{Age} \in [11, 25] \wedge \mathsf{Income} \in [11, 25]$, and we can decompose it into minimum nine nodes in four different ways, *i.e.*, $\mathcal{I}_1, \ldots, \mathcal{I}_4$, without overlapping.

For decomposition $\mathcal{I}^{\mathsf{C}}$ of C, with $|\mathcal{I}^{\mathsf{C}}|$ nodes, since the frequencies of these nodes are added together for C, and the error bound for each node is the same, we have:

$$\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{C}}(\mathbf{y})) \propto |\mathcal{I}^{\mathsf{C}}|. \tag{10}$$

Thus, the optimal decomposition of C is the one with the minimum number of nodes among all possible decompositions. The possible decompositions include the ones that only add up the nodes, as in previous work [32, 8], as well as those that subtract some nodes to achieve the equivalent range conditions, *e.g.*, $\mathcal{I}_4$ in Figure 6.

To find the decomposition with minimum number of nodes, we can first find the optimal decomposition for each single-attribute condition of C, and cross product them into the optimal decomposition of C. For the example in Figure 6, ① we find the top-2 decompositions for attributes Age and Income separately, and ② cross-product them to get the top-4 decompositions for C. For ①, we can solve for the top-$k$ decompositions for all possible ranges of each individual attribute using dynamic programming offline, with complexity polynomial to the domain size.

## 6.2 Consistency Optimization

The top-$k$ decompositions of C are *consistent*, *i.e.*, they are all unbiased estimations of C, and we can combine them for better utility. For the top-$k$ decompositions of C with $|\mathcal{I}^{\mathsf{C}}_1|, \ldots, |\mathcal{I}^{\mathsf{C}}_k|$ nodes, respectively, we assign $w_i$ as the weight for $\mathcal{I}^{\mathsf{C}}_i$, and the weighted average of the $k$ decompositions has $\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{C}}(\mathbf{y})) \propto \sum_{i=1}^{k} w_i^2 |\mathcal{I}^{\mathsf{C}}_i|$. We can minimize the error bound using the KKT condition, under the constraint that $\sum_{i=1}^{k} w_i = 1$, and the optimal weights are $w_i = (1/|\mathcal{I}^{\mathsf{C}}_i|)/(\sum_{j=1}^{k} 1/|\mathcal{I}^{\mathsf{C}}_j|)$. Hence, the optimal error bound of the weighted average of the top-$k$ decompositions is

$$\mathrm{MSE}(\bar{\mathbf{P}}^{\mathsf{C}}(\mathbf{y})) \propto 1/(\textstyle\sum_{i=1}^{k} 1/|\mathcal{I}^{\mathsf{C}}_i|), \tag{11}$$

which can be as small as $\frac{1}{k}$ of Equation (10), *i.e.*, when the top-$k$ decompositions all have the same number of nodes. We state the utility improvement in Lemma 9

**Lemma 9.** For range frequency aggregation of C, weighted averaging its top-$2^{d_q}$ decompositions can improve the utility by $2^{d_q}$.

**Proof Sketch:** Let's first consider the simple case where each hierarchy of the $d_q$ range attributes is single layer with $B$ leaves. Then, suppose the optimal decomposition has $l_i$ leaves on the $i$-th attribute, and the total number of nodes of the decomposition is $\prod_{i=1}^{d_q} l_i$. For the $i$-th attribute, we can substitute its $l_i$ leaves with $B - l_i + 1$ intervals, *i.e.*, the parent interval minus the other $B - l_i$ leaves. In addition, the substitution does not overlap with the original optimal decomposition. Thus, we have $2^{d_q}$ consistent decompositions by choosing either the optimal or its substitution on each attribute. The error bound of the optimal decomposition is $\propto \prod_{i=1}^{d_q} l_i$. And the error bound of the weighted average of the $2^{d_q}$ consistent decompositions is

$$\propto \prod_{i=1}^{d_q} l_i/(1 + \sum_{j=1}^{d_q} \sum_{1 \leqslant i_1 < \cdots < i_j \leqslant d_q} \prod_{k=1}^{j} \frac{B - l_{i_k} + 1}{l_{i_k}}).$$

When $l_i = B - l_i + 1$, *i.e.*, $l_i = \frac{B+1}{2}$, the error bound is $2^{-d_q}$ of that of the single optimal decomposition.

For hierarchies with more than one layers, the above argument applies, and the improvement reaches $2^{d_q}$ when there is a non-overlapping substitution with equal number of intervals for the optimal decomposition on each individual attribute in C. □

We evaluate the effect of $k$ on the aggregation utility in Section 7.

## 7. EVALUATION

We evaluate the utility of our end-to-end framework using synthetic and real-world datasets and queries. We conduct the evaluation on an Intel Xeon Platinum 8163 2.50GHz PC with 64GB memory. We set up a single node Spark cluster on the machine, using Hadoop 2.7.1, Spark 2.4.3 and Hive 2.3.5. We register the $\mathcal{R}$'s and $\bar{\mathbf{P}}$'s of our mechanisms as UDF's of SparkSQL. Each dataset is first loaded as table into Spark SQL, and we perturb each row, as a tuple, using the perturb UDF and collect them as the table to be released. At query time, ordinary SQL statement is issued from the user interface, and automatically rewritten using the proper $\bar{\mathbf{P}}$ UDF's, which will be executed by the underlying SparkSQL engine on the released table. For joint aggregation, multiple released tables are first joined on the join key by SparkSQL, and the SQL engine applies the joint frequency oracles over joined tuples and adds up their contributions into the result. We will open-source the code and the platform setup as a docker container.

**Dataset** We evaluate with two synthetic and two real-world datasets:

- SYN-1: Single-table synthetic data with 4 ordinal and 4 categorical attributes, and 1 non-sensitive ordinal attributes. The number of tuples ranges from $2^{-2} \times 10^6$ to $2^2 \times 10^6$ (default $1 \times 10^6$). The domain size $m$ for ordinal ranges from $5^2$ to $5^4$ (default $5^3$), and that for categorical is 500. We simulate correlations among ordinal attributes by sampling from normal distribution, with ($\mu = \frac{m}{2}, \sigma = \frac{m}{4}$) for one of them, and adding it with Gaussian noises ($\mu = 0, \sigma = 10$) for others.
- SYN-2: Synthetic data of two tables, both with one ordinal and one categorical attributes. For ordinal, $m = 5^3$, and, for categorical, $m = 500$. The two tables can be joined by the forged $id$, with two configurations: i) *one-to-one*; and ii) *one-to-many*, where a tuple in table one has $[1, 10]$ matching tuples in table two. And we test with $1 \times 10^6$ and $2 \times 10^6$ joined tuples.
- PUMS-P *[29]*: $3 \times 10^6$ census records of US citizens in 2017, with attributes AGE, MARST and UHRSWORK.
- PUMS-H *[29]*: 1940 census records of US households, with attributes STATE and CITY, and citizens, with attributes SEX, AGE, RACE and INCOME. Both records have HID as the join key, and it is primary-key for household and foreign-key for citizens. We focus on two samples: *IN*): $3.42 \times 10^6$ joined records from Indiana; and *IL*): $7.56 \times 10^6$ joined records from Illinois. For both samples, each household has up to $\tau_{\max} = 10$ citizens.

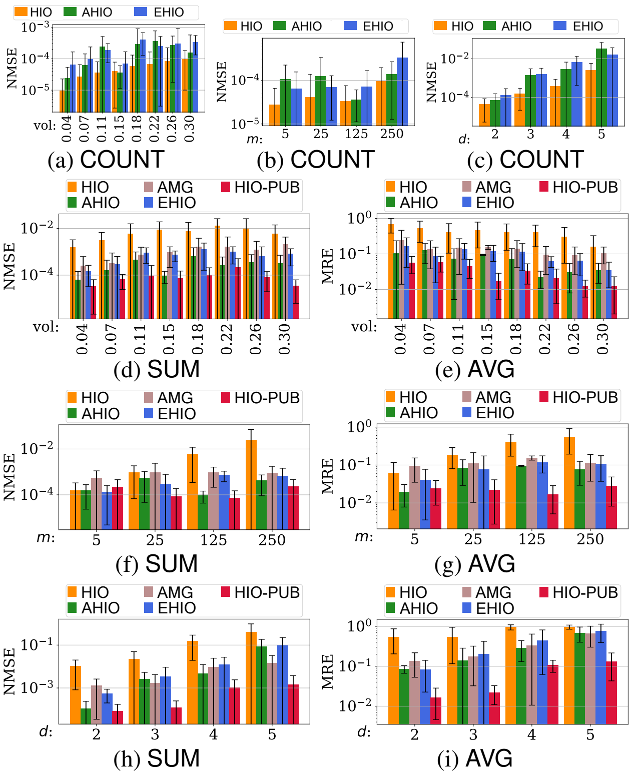


Figure 7: Aggregation on sensitive attribute using SYN-1.

**Mechanisms.** We consider six mechanisms in the evaluation:

- HIO: The mechanism proposed in [32] for single-table MDA.
- SC-JOIN: Joint aggregation mechanism using SC (Section 4.1).
- AHIO: The augment-then-perturb instantiation of PRP (Section 3.3).
- EHIO: The embed-then-perturb instantiation of PRP (Section 3.3).
- HIO-JOIN: Joint aggregation over HIO (Section 4.2), with augment-then-perturb and $\tau$-truncation (Section 5.2).
- *-RDC: Mechanism * with utility optimization (Section 6).

We set the fan-out as 5 for hierarchies in both HIO and SC.

**Queries.** We evaluate the utility of aggregation queries for COUNT, SUM and AVG, as well as their sensitivity to factors: i) $m$: the domain size of the aggregation attribute; ii) vol: the volume of the predicate, *i.e.*, the ratio of the predicate space over the entire domain; and iii) $d_q$: the number of attributes in the predicate.

**Metrics.** We use two metrics to evaluate the estimation utility of $\bar{\mathbf{P}}$ over a set $\mathcal{Q}$ of queries of the same characteristics,

- *Normalized Mean Squared Error.* It measures how large the errors are relative to the maximum possible answer, *i.e.*, $\mathrm{NMSE}(\bar{\mathbf{P}}(\mathcal{Q})) = \frac{1}{|\mathcal{Q}|} \sum_{\mathsf{Q} \in \mathcal{Q}} \left( (\bar{\mathbf{P}}^{\mathsf{Q}}(\mathcal{R}(\mathrm{T})) - \mathbf{P}^{\mathsf{Q}}(\mathrm{T}))/\Sigma_{\mathrm{T}} \right)^2$ where $\Sigma_T = |T|$ for COUNT, and $\Sigma_T = \sum_{t \in T} |t[A]|$ for SUM, are the upper bounds of aggregation. Note that this metric is identical to the mean square error used in [8].
- *Mean Relative Error.* It measures how large the errors are relative to the true answers, *i.e.*, $\mathrm{MRE}(\bar{\mathbf{P}}(\mathcal{Q})) = \frac{1}{|\mathcal{Q}|} \sum_{\mathsf{Q} \in \mathcal{Q}} \left| (\bar{\mathbf{P}}^{\mathsf{Q}}(\mathcal{R}(\mathrm{T})) - \mathbf{P}^{\mathsf{Q}}(\mathrm{T}))/\mathbf{P}^{\mathsf{Q}}(\mathrm{T}) \right|$.

We use NMSE for COUNT and SUM, and MRE for AVG.

| $\epsilon =$ | | 0.5 | 1.0 | 2.0 | 5.0 | True |
|---|---|---|---|---|---|---|
| Q1 | HIO | 32.74<br>$\pm 20.03$ | 28.25<br>$\pm 6.46$ | 27.49<br>$\pm 4.07$ | 26.6<br>$\pm 0.69$ | 26.62 |
| | AHIO | 26.88<br>$\pm 5.15$ | 27<br>$\pm 2.82$ | 26.71<br>$\pm 1.2$ | 26.23<br>$\pm 0.75$ | |
| | EHIO | **26.93**<br>**$\pm 4.07$** | **26.76**<br>**$\pm 2.06$** | **26.07**<br>**$\pm 1.05$** | **26.7**<br>**$\pm 0.27$** | |
| | AHIO -RDC | 26.88<br>$\pm 5.15$ | 27<br>$\pm 2.82$ | 26.71<br>$\pm 1.2$ | 26.23<br>$\pm 0.75$ | |
| Q2 | HIO | 44.6<br>$\pm 47.24$ | 35.95<br>$\pm 28.81$ | 33.91<br>$\pm 10.08$ | 30.73<br>$\pm 3.36$ | 29.98 |
| | AHIO | 24.53<br>$\pm 12.08$ | 24.33<br>$\pm 5.79$ | 29.63<br>$\pm 3.43$ | 29.55<br>$\pm 1.25$ | |
| | EHIO | 28.60<br>$\pm 18.71$ | 29.16<br>$\pm 7.2$ | 29.33<br>$\pm 2.89$ | 30<br>$\pm 0.7$ | |
| | AHIO -RDC | **31.06**<br>**$\pm 10.37$** | **29.64**<br>**$\pm 4.71$** | **29.58**<br>**$\pm 1.82$** | **29.95**<br>**$\pm 0.69$** | |
| Q3 | HIO | $-110.27$<br>$\pm 320.65$ | 22.76<br>$\pm 121.81$ | 54.35<br>$\pm 73.6$ | 26.32<br>$\pm 8.48$ | 30.82 |
| | AHIO | 4.15<br>$\pm 133.45$ | 37.5<br>$\pm 67.08$ | 26.21<br>$\pm 20.34$ | 29.57<br>$\pm 4.91$ | |
| | EHIO | 42.58<br>$\pm 243.41$ | 26.58<br>$\pm 36.65$ | 29.18<br>$\pm 14.62$ | 30.78<br>$\pm 1.76$ | |
| | AHIO -RDC | **25.13**<br>**$\pm 98.24$** | **30.7**<br>**$\pm 25.03$** | **29.18**<br>**$\pm 9.12$** | **30.83**<br>**$\pm 1.74$** | |

Figure 8: Q1-Q3 on PUMS-P.

## 7.1 Attribute Aggregation

We first evaluate aggregation on attribute, with COUNT, SUM and AVG queries, using the HIO, AHIO and EHIO mechanisms.

**Benchmark with SYN-1.** First, we evaluate the effect of the volume of the aggregation query. We sample queries with range volume from 0.04 up to 0.30, and fixed $\epsilon = 2.0$, $d = 2$, $m = 125$ and $d_q = 1$. Figures 7a, 7d, and 7e show the aggregation errors for HIO, AHIO and EHIO. We observe that, for COUNT, HIO performs better than AHIO and EHIO, and the reason is that both AHIO and EHIO spare privacy budget for the rounding value to support attribute aggregation. For SUM and AVG, AHIO and EHIO outperform HIO consistently. The volume does not have much effect on COUNT or SUM, and the relative error of AVG decreases as the volume increases.

Second, we evaluate the effect of $m$, and test with $m = 5$, 25, 125 and 250 for the aggregation attribute, with $\epsilon = 2.0$, $d = 2$, vol $= 0.15$, and $d_q = 1$ (Figures 7b,7f and 7g). We first observe that the domain size has slight effects on the absolute error of COUNT since there is an logarithmic relation between the hierarchy height and the domain size. As for SUM and AVG, the domain size does not affect the errors for AHIO or EHIO much, and the error for HIO increases fast as the domain size increases.

Third, we evaluate the effect of the number of attributes, *i.e.*, $d$, in the data, and test with $d = 2$, 3, 4 and 5 (Figures 7c, 7h and 7i). As $d$ increases, all the errors increase.

To better understand the effect of the rounding technique, we evaluate using HIO when the aggregation attribute is released as non-sensitive. We show its results as HIO-PUB in Figure 7, and we only report for SUM and AVG since its utility on COUNT is the same as HIO. We observe consistent drop on utility for AHIO and EHIO, compared to HIO-PUB.

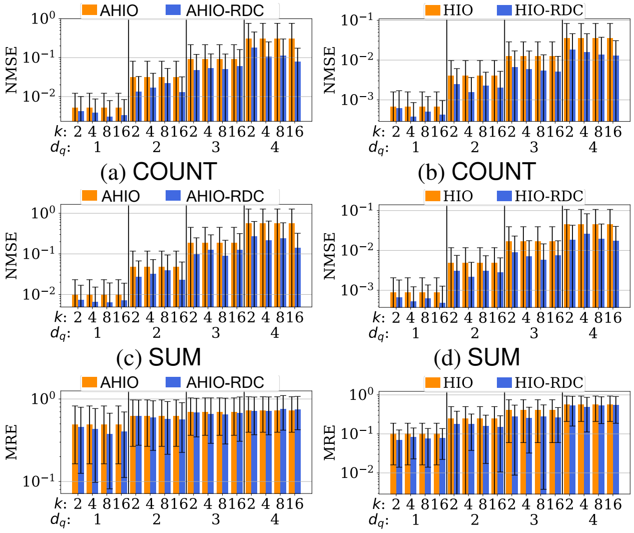
We also evaluate the PRP framework when the underlying multi-dimensional analytical frequency oracle is *marginal release* (MG) [11], and we report the results for augment-then-perturb using MG (AMG) in Figure 7. As the volume increases, the aggregation error of AMG increases much faster than those of AHIO and EHIO (Figure 7d and 7e) because MG estimates a large range predicate with many point conditions. When the volume is not large, *e.g.*, $\leqslant 0.15$, the errors of AMG are comparable to those of AHIO and EHIO.

**Sample Queries on PUMS-P.** We test on the PUMS-P dataset with the following three queries:

Q1: SELECT AVG(UHRSWORK) FROM PUMS-P WHERE MARST = Married;

Q2: SELECT AVG(UHRSWORK) FROM PUMS-P WHERE MARST = Married AND $31 \leqslant$ AGE $\leqslant 70$;

Q3: SELECT AVG(UHRSWORK) FROM PUMS-P WHERE MARST = Single AND $31 \leqslant$ AGE $\leqslant 50$.



(a) COUNT (b) COUNT

(c) SUM (d) SUM

(e) AVG (f) AVG

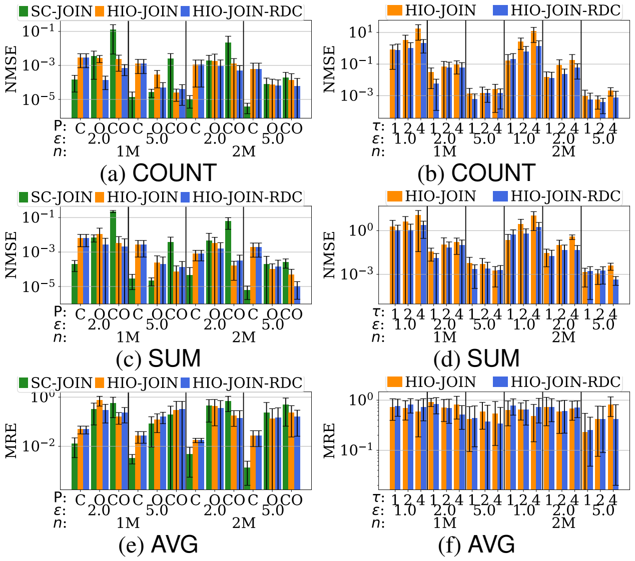
Figure 9: Effect of range consistency optimization using SYN-1. Left: aggregation on sensitive attribute. Right: aggregation on non-sensitive attribute. $k$ indicates the number of top decompositions and $d_q$ indicates the number attributes in the range predicate.

We set $\epsilon = 0.5, 1, 2, 5$, and test with HIO, AHIO, EHIO and AHIO-RDC. For each setting, we release the data 10 times, evaluate Q1-Q3 on the 10 releases, and report the mean aggregation results, together with the standard deviations, in Figure 8. We include the true results for the three queries in the right-most column. For all mechanisms and settings, *i.e.*, the pair of query and $\epsilon$, we have the true aggregation results in the standard deviation intervals. For each setting, we highlight the mechanism with the smallest standard deviation because it provides the best confidence interval. We observe that AHIO, EHIO and AHIO-RDC consistently out-performs HIO, and AHIO and EHIO are comparable. AHIO-RDC performs better than AHIO, especially for the more selective queries, *e.g.*, Q3. In addition, the standard deviation increases when the range predicate gets more selective, and decreases as $\epsilon$ increases.

## 7.2 Range Consistency Optimization

We next evaluate the effect of the range consistency optimization on attribute aggregations using the AHIO and AHIO-RDC mechanisms. We use SYN-1 with 4 ordinal sensitive attributes, and fix $\epsilon = 2$. We evaluate the effects of $d_q$ and the number of top decomposition $k$. In particular, we evaluate aggregation queries with range predicate on $d_q = 1$, 2, 3 and 4 of the attributes, and vary $k$ to be 2, 4, 8 and 16. Figures 9a, 9c and 9e show the results. We observe that HIO-RDC improves the utility for COUNT and SUM, as the number of sensitive attributes in the range predicate increases. In addition, larger $k$ performs better when the number of attributes in the predicate is larger, which is consistent with Equation (11). Note that the improvement on utility is not exactly $k$ because the analysis in Lemma 9 is the upper bound when the $k$ decompositions are of the same error bound, which is not guaranteed for randomly selected range predicates in our experiments.

To benchmark the effectiveness of the range consistency optimization against the state of the art, we compare the utility of HIO-RDC against that of HIO, using queries that aggregate on the non-sensitive attribute of SYN-1 with range predicate on 1, 2, 3 and 4



(a) COUNT

(b) COUNT

(c) SUM

(d) SUM

(e) AVG

(f) AVG

Figure 10: Joint aggregation over SYN-2. Left: one-to-one, C, O, CO indicate one categorical, one ordinal, one categorical and one ordinal attributes in the predicate, respectively. Right: one-to-many, $\tau$ is the truncation number.

of the ordinal attributes, and $k \in \{2, 4, 8, 16\}$. Figures 9b, 9d, and 9f show the results, and HIO-RDC consistently outperforms HIO for all the three types of aggregations.

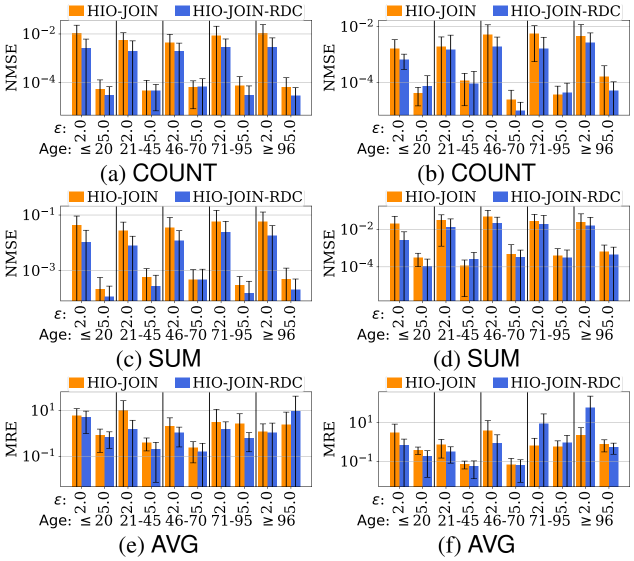
## 7.3 Joint Aggregation

Finally, we evaluate joint aggregation across two tables. We first evaluate with SC-JOIN, HIO-JOIN and HIO-JOIN-RDC using SYN-2 for sensitivity analysis. Then we evaluate with HIO-JOIN and HIO-JOIN-RDC on PUMS-H as case-study.

**One-to-one.** We test with queries of different range predicates on the attributes: i) C: one point condition; ii) O: one range condition; and iii) CO: one point and one range conditions. The volume for the range condition is fixed at 0.12. If the predicate involves only one attribute, then the aggregation attribute is in the other table. Figures 10a, 10c and 10e show the results. First, the overall estimation utility improves as either $\epsilon$ or the table size $n$ increases. Second, as the predicate gets complicated, HIO-JOIN outperforms SC-JOIN. In addition, HIO-JOIN-RDC consistently improves the aggregation utility over HIO-JOIN.

**One-to-many.** We test with queries that aggregate on the attribute with the CO range predicate of volume 0.12. We evaluate the HIO-JOIN and HIO-JOIN-RDC schemes because, as we show above, SC-JOIN is worse than HIO-JOIN for this kind of range predicate. For the one-two-many setting, we enforce the user-level LDP using $\tau$-truncation, and we evaluate the same aggregation with $\tau = 1, 2, 4$. Figures 10b, 10d and 10f show the results. First, larger $\epsilon$ or table size $n$ leads to better estimation utility for all types of queries. Second, as $\tau$ increases, the estimation utility on COUNT and SUM decreases, and the effect on AVG is not substantial. Third, the range optimization technique improves the overall utility consistently.

PUMS-H **Case Study.** We conduct a case study using the PUMS-H dataset, for Indiana and Illinois, respectively. The study answers the following question: *how many people living in Indiana (Illinois) are in the city Indianapolis (Chicago) and in the specific age group?* In addition to the count, we include in the analysis the sum and average on AGE. The age groups are [1-20], [21-45], [46-70], [71-95] and [96-120]. We evaluate HIO-JOIN and HIO-JOIN-RDC, and set $\epsilon = 2$ and 5. The truncation number $\tau$ is fixed at 1 for this study. Figure 11a, 11c and 11e show the results for Indiana, and Figure 11b, 11d and 11f for Illinois.



(a) COUNT

(b) COUNT

(c) SUM

(d) SUM

(e) AVG

(f) AVG

Figure 11: Joint aggregation over PUMS-H. Left: Indiana. Right: Illinois. The predicate is "CITY $= x$ AND AGE $\in [l, r]$".

# 8. EXTENSIONS AND DISCUSSION

## 8.1 Handling Group-by Queries

Our solution can be extended for *group-by queries* to perform aggregation analysis over joins of relations from different services and summarize the results by some *group-by attribute*. To answer the group-by queries, we share the same assumption, as previous works [13, 25, 1] do, that the dictionary (*i.e.*, the set of all possible values) of the group-by attribute is known to public. Note that this assumption does not affect the privacy of each individual's data, as the dictionary is independent on the content of the data and is usually public knowledge (*e.g.*, group by CITY or RACE).

**Algorithm** HIO-GROUP-BY**.** Consider a query Q $= (A, \mathsf{C}, T)$ aggregating $A$ under predicate C on a relation or join of relations $T$, with a group-by attribute G. Abusing the notation, we also use G to denote the dictionary of attribute G. With the perturbation algorithms ($\tau$-Truncation and Partition-Rounding-Perturb framework to ensure $\epsilon$-uLDP) unchanged, our query estimation algorithm HIO-GROUP-BY is outlined in Figure 12. Let consider two cases:

If G is a nonsensitive attribute and known to public, we can partition the perturbed relation $\mathcal{R}(T)$ by G. For each group $v \in$ G, we read the perturbed tuples of this group into $T(v)$ (line 3) and apply HIO-JOIN on $T(v)$ to estimate the aggregation value (line 4).

If G is a sensitive attribute, we can still enumerate all possible values of G with the public dictionary, but cannot access its true value for each perturbed tuple in $\mathcal{R}(T)$. To estimate the answer to Q, we rewrite it a bit: for each group $v \in$ G, we extend the predicate C to be "C $\wedge$ G $= v$" and apply HIO-JOIN to process Q with the extended predicate on $\mathcal{R}(T)$ (line 5). In this way, we obtain an unbiased estimate of the aggregation value for each group.

**Error bounds.** If G is non-sensitive, the error bound of estimated aggregation in each group follows from Lemmas 4, 5, or 8 for different join types. As each group is processed independently, the mean squared error (MSE) is proportional to the size of the group ($n$ in the lemmas is equal to the number of tuples in each group).

The case when G is sensitive is more difficult. The error bound for different join types again follows from Lemmas 4, 5, or 8. However, since the aggregation value is recovered from all the perturbed tuples with an additional constraint "G $= v$", the MSE is proportional to the total number of tuples in $T$ ($n$ in the lemmas is equal

```
Aggregation query $\mathsf{Q} = (A, \mathsf{C}, T)$ group by $\mathsf{G}$:
for $v \in \mathsf{G}$ do
  if $\mathsf{G}$ is nonsensitive then
    $T(v) \leftarrow \sigma_{\mathsf{G}=v}(\mathcal{R}(T))$   ($\sigma.(\cdot)$ is selection)
    $S_v \leftarrow \bar{\mathbf{P}}^{(A,\mathsf{C})}_{\text{HIO-JOIN}}(T(v))$
  else $S_v \leftarrow \bar{\mathbf{P}}^{(A,\mathsf{C}\wedge\mathsf{G}=v)}_{\text{HIO-JOIN}}(\mathcal{R}(T))$
return $S$
```

Figure 12: HIO-GROUP-BY: for group-by queries

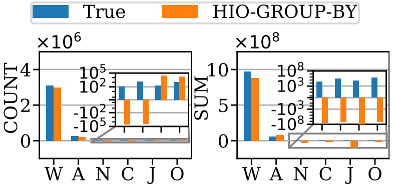


Figure 13: One run of HIO-GROUP-BY estimation versus the ground truth.

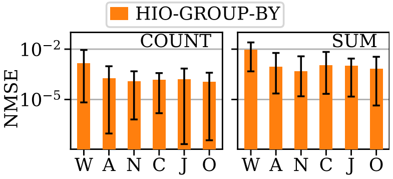


Figure 14: Average errors (over 15 runs) of HIO-GROUP-By by group.

to $|T|$) as well as the aggregation value. Note that MSE is the "squared" error, the above error bound implies that, for groups with large aggregation values, the relative estimation errors are smaller.

**Case study and empirical evaluation.** We test a group-by query on the PUMS-H dataset: what are COUNT and SUM(Income) of people whose households are in Chicago, grouped by RACE[1]? All the attributes, including RACE, are sensitive. We compare the estimates obtained by HIO-GROUP-BY in one run ($\epsilon = 5$) with the ground truth (Figure 13), and further report the average error (NMSE) of each group for 15 runs of the mechanism (Figure 14). The key observation is that, our HIO-GROUP-BY preserves the trend across groups very well especially for large groups (*e.g.*, groups 'W' and 'A'), which enables us to identify the "top groups" and support accurate decision making on them. NMSE for all the groups are very close, which is consistent with the theoretical error bounds. For small groups (*e.g.*, groups 'J' and 'O'), the error is relatively large because of small aggregation values, which is an inherently difficult case for DP-based estimations.

## 8.2 Join with Star Schema

We focus on two-table primary-foreign-key join in Section 5, and can extend to join with the more complex star schema, where one service collects tuples with foreign keys to multiple primary-key tuples collected by different services. Such schema is common in business data warehouse. For instance, in Example 1.1, a third service could collect tuples on products, with attributes Price and Country, and a unique product id PID. The collected transaction tuples further contain the foreign key PID of the corresponding product. And an analyst wants to know the total sales from certain users on products from certain country, *e.g.*,

```
SELECT SUM(Amount) FROM Transaction
JOIN Product ON Transaction.PID = Product.PID
JOIN User ON Transaction.UID = User.UID
WHERE Country = "China" AND Age ∈ [20, 30].
```

We can extend $\tau$-truncation to handle such relation, under $\epsilon$-uLDP. The major adaptation is that we need to enforce that the same number of tuples with foreign-key, *i.e.*, transaction, match each pair of primary-key tuples, *i.e.*, ⟨user,product⟩. Thus, for $n$ users and $n$ products, with $\tau$-truncation, we need to collect $n^2\tau$ transaction tuples. Thus, for joint aggregation, we can join the perturbed values from the three services, and aggregate on the joined values. It is possible to optimize such straight-forward extension for better efficiency and privacy management, which we leave as future work.

# 9. RELATED WORK

We review related topics in both the centralized setting of differential privacy (DP) [14] and its local model (LDP).

**LDP mechanisms.** There have been several LDP frequency oracles [15, 5, 4, 31, 2] proposed. They rely on techniques like hashing (*e.g.*, [31]) and Hadamard transform (*e.g.*, [4, 2]) for good utility. LDP mean estimation is another basic task [12, 30] with stochastic rounding as a subroutine. Frequency oracles are also used in other tasks, *e.g.*, finding heavy hitters [4, 34, 6], frequent itemset mining [27, 33], and marginal release [28, 7, 36].

**Answering range queries.** Range counting queries are supported in the centralized setting of DP via, *e.g.*, hierarchical intervals [17] (one-dim range queries) or via wavelet [35] (multi-dimensional range queries). [26] optimizes the hierarchical intervals in [17] by choosing a proper branching factor. McKenna *et al.* [23] propose a method to collectively optimize errors in high-dimensional queries of a given workload under the centralized setting of DP. It will be interesting to design such workload-dependent techniques under LDP.

In both marginal release and range queries, it has been noticed that constrained inference could boost the accuracy while enforcing the consistency across different marginal tables and intervals (*e.g.*, [3, 17, 10, 26]). Since it is a post-processing step, all the consistency enforcement techniques in the centralized setting of DP can be potentially used in the LDP setting.

**Joint analysis in LDP.** Several methods have been proposed to handle the joint analysis in LDP. In particular, [16] proposed to use EM algorithm. The starting point is to find on that maximizes the likelihood (possibility) of the observed report. Later, [34] derived formulas to direct evaluate the joint estimation, which essentially extends the aggregation function to multi-dimensional setting. Both methods work in the categorical setting. For the ordinal setting [32] proposed a method based on matrix inversion.

**Frequent itemset mining in LDP.** Existing work [27, 33] on LDP frequent itemset mining addresses the problem of identifying the frequent items from $n$ users, each generates a subset of values from the domain $\mathbb{D}$, with $\epsilon$-LDP. They share this core technique called *padding-and-sampling*, which pads the set of values of each user to a fixed size $\ell$, and samples one to perturb and report. [33] further leverages the privacy amplification from sampling to boost the utility. The key difference between our $\tau$-truncation and the padding-and-sampling is that $\tau$-truncation does not need padding because it enables unbiased estimation over tuples with distinct sampling ratios, and padding causes biased under-estimation (small $\ell$) or amplified error bound (large $\ell$).

**SQL support in DP.** In the centralized model of DP, where there is a *trusted party*, there are efforts to support SQL queries, including PINQ [24], wPINQ [25], Flex [18], and PrivateSQL [20]. These systems assume a trusted data engine [21] that maintains users' exact data, and injects noise during the (offline or online) analytical process so that query results transferred across the firewall ensures DP. Different from that, our paper assumes no such trusted party.

# 10. CONCLUSION

In this work, we focus on collecting and analyzing data jointly from multiple services under local differential privacy. We introduce the notation of user-level LDP to formalize and protect the privacy of a user when her joined tuples are released. We propose mechanisms and estimation methods to process multi-dimensional analytical queries, each with attributes (in its aggregation and predicates) collected and perturbed independently by multiple services.

[1] W: White; A: African American; N: American Indian or Alaska Native; C: Chinese; J: Japanese; O: Other Asian or Pacific Islander.

# 11. REFERENCES

[1] Learning with privacy at scale. *Apple Machine Learning J.*, 2017.

[2] J. Acharya, Z. Sun, and H. Zhang. Hadamard response: Estimating distributions privately, efficiently, and with little communication. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1120–1129, 2019.

[3] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*, 2007.

[4] R. Bassily, K. Nissim, U. Stemmer, and A. G. Thakurta. Practical locally private heavy hitters. In *NIPS*, 2017.

[5] R. Bassily and A. D. Smith. Local, private, efficient protocols for succinct histograms. In *STOC*, 2015.

[6] M. Bun, J. Nelson, and U. Stemmer. Heavy hitters and the structure of local privacy. In *PODS*, 2018.

[7] G. Cormode, T. Kulkarni, and D. Srivastava. Marginal release under local differential privacy. In *SIGMOD*, 2018.

[8] G. Cormode, T. Kulkarni, and D. Srivastava. Answering range queries under local differential privacy. *PVLDB*, 2019.

[9] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In *NIPS*, 2017.

[10] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *SIGMOD*, 2011.

[11] Z. Ding, Y. Wang, G. Wang, D. Zhang, and D. Kifer. Detecting violations of differential privacy. In *CCS*, 2018.

[12] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *FOCS*, 2013.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

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

[15] Ú. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.

[16] G. Fanti, V. Pihur, and Úlfar Erlingsson. Building a rappor with the unknown: Privacy-preserving learning of associations and data dictionaries. *Proceedings on Privacy Enhancing Technologies*, 2016.

[17] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1), 2010.

[18] N. M. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for SQL queries. *PVLDB*, 11(5), 2018.

[19] M. Joseph, A. Roth, J. Ullman, and B. Waggoner. Local differential privacy for evolving data. In *NeurIPS*, 2018.

[20] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. Privatesql: A differentially private sql query engine. *PVLDB*, 2019.

[21] I. Kotsogiannis, Y. Tao, A. Machanavajjhala, G. Miklau, and M. Hay. Architecting a differentially private SQL engine. In *CIDR*, 2019.

[22] Z. Li, T. Wang, M. Lopuhaä-Zwakenberg, B. Skoric, and N. Li. Estimating numerical distributions under local differential privacy. In *SIGMOD*, 2020.

[23] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *PVLDB*, 11(10), 2018.

[24] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, 2009.

[25] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis. *PVLDB*, 2014.

[26] W. H. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 6(14), 2013.

[27] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren. Heavy hitter estimation over set-valued data with local differential privacy. In *CCS*, 2016.

[28] X. Ren, C. Yu, W. Yu, S. Yang, X. Yang, J. A. McCann, and P. S. Yu. Lopub: High-dimensional crowdsourced data publication with local differential privacy. *IEEE Trans. Information Forensics and Security*, 13(9), 2018.

[29] S. Ruggles, J. T. Alexander, K. Genadek, R. Goeken, M. B. Schroeder, and M. Sobek. Integrated public use microdata series: Version 5.0 [machine-readable database], 2010.

[30] N. Wang, X. Xiao, Y. Yang, T. D. Hoang, H. Shin, J. Shin, and G. Yu. Privtrie: Effective frequent term discovery under local differential privacy. In *ICDE*, 2018.

[31] T. Wang, J. Blocki, N. Li, and S. Jha. Locally differentially private protocols for frequency estimation. In *USENIX Security*, 2017.

[32] T. Wang, B. Ding, J. Zhou, C. Hong, Z. Huang, N. Li, and S. Jha. Answering multi-dimensional analytical queries under local differential privacy. In *SIGMOD*, 2019.

[33] T. Wang, N. Li, and S. Jha. Locally differentially private frequent itemset mining. In *SP*, 2018.

[34] T. Wang, N. Li, and S. Jha. Locally differentially private heavy hitter identification. *IEEE Trans. Dependable Sec. Comput.*, 2019.

[35] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, 2010.

[36] Z. Zhang, T. Wang, N. Li, S. He, and J. Chen. Calm: Consistent adaptive local marginal for marginal release under local differential privacy. In *CCS*, 2018.