DPSAaS: Multi-Dimensional Data Sharing and Analytics as Services under Local Differential Privacy

Min Xu[2]* Tianhao Wang[3]† Bolin Ding[1] Jingren Zhou[1] Cheng Hong[1] Zhicong Huang[1]
[1]Alibaba Group [2]University of Chicago [3]Purdue University
{bolin.ding, jingren.zhou, vince.hc, zhicong.hzc}@alibaba-inc.com,
xum@cs.uchicago.edu, tianhaowang@purdue.edu

ABSTRACT

Differential privacy has emerged as the *de facto* standard for privacy definitions, and been used by, *e.g.*, Apple, Google, Uber, and Microsoft, to collect sensitive information about users and to build privacy-preserving analytics engines. However, most of such advanced privacy-protection techniques are not accessible to mid-size companies and app developers in the cloud. We demonstrate a lightweight middleware **DPSAaS**, which provides **d**ifferentially **p**rivate data-**s**haring-and-**a**nalytics functionality **a**s cloud **s**ervices.

We focus on multi-dimensional analytical (MDA) queries under local differential privacy (LDP) in this demo. MDA queries against a fact table have predicates on (categorical or ordinal) dimensions and aggregate one or more measures. In the absence of a trusted agent, sensitive dimensions and measures are encoded in a privacy-preserving way locally using our LDP data sharing service, before being sent to the data collector. The data collector estimates the answers to MDA queries from the encoded data, using our data analytics service. We will highlight the design decisions of DPSAaS and twists made to LDA algorithms to fit the design, in order to smoothly connect DPSAaS to the data processing platform and analytics engines, and to facilitate efficient large-scale processing.

PVLDB Reference Format:
M. Xu, T. Wang, B. Ding, J. Zhou, C. Hong, and Z. Huang. DPSAaS: Multi-Dimensional Data Sharing and Analytics as Services under Local Differential Privacy. *PVLDB*, 12(12): 1862-1865, 2019.
DOI: https://doi.org/10.14778/3352063.3352085

1. INTRODUCTION

Informed business decisions can be made from large volumes of data about user profiles and activities. In order to meet users' expectation of their privacy, rigorous privacy guarantees need to be provided to them on how their sensitive data is collected, shared, and analyzed. To this end, the *de facto* privacy standard, differential privacy (DP) [4], is being used by, *e.g.*, Apple [1], Google [5], Uber [6], and Microsoft [2]. Informally, differential privacy requires that the output of a data sharing or analytics process varies little with any change in an arbitrary individual's sensitive value.

The centralized DP model assumes that a *trusted* data collector maintains exact data from users, and injects noise in the analytical process to guarantee DP. For example, the solution by Uber [6], as well as PINQ and wPINQ [8, 9], and PrivSQL [7], to answer SQL queries under DP is built on a SQL engine which is trustable, *e.g.*, internally in Uber, and injects DP noise into query results.

In the absence of a trusted party, users prefer not to have their sensitive data leave their devices or workspaces in an unprotected form, and thus, the centralized DP model is no longer applicable. In such scenarios, one (*e.g.*, enterprises [1, 5, 2]) can adopt the *local differential privacy* model (LDP) [3]. Each user's sensitive data is encoded by a randomized algorithm before being sent to the data collector. LDP guarantees that the likelihood of any specific output of the algorithm varies little with input, *i.e.*, the sensitive data. In this way, users do not need to trust the data collector.

An application scenario. LDP fits the class of analytical applications in this demo well. Suppose a number of individuals use an online shopping app. Users are anonymous. Each generates *multi-dimensional data* as in Table 1. Age, Salary, and State are *sensitive attributes*, and the rest are *non-sensitive*. Some attributes are *measures* to be aggregated in analytics, *e.g.*, Salary, ActiveTime (how much time a user spent in the app) and Purchase. *Data owners* (users in this example) prefer to have their sensitive data shared with a *data collector* (the app server) in a privacy-preserving way, *e.g.*, under LDP. Note that the data collector is not trusted, and thus the privacy has to be preserved before data leaves each owner's device or workspace. On the other side, the data collector wants to analyze how the app performs by issuing analytical queries, *e.g.*,

$$\text{Q\_SUM} = \texttt{SELECT SUM(Purchase) FROM } T \quad (1)$$
$$\texttt{WHERE Age} \in [30, 40] \texttt{ AND Salary} \in [50\text{K}, 150\text{K}],$$

which aggregates Purchase under constraints on sensitive attributes.

To this end, our paper that appears in SIGMOD 2019 [10] studies how to (approximately) answer a class of *multi-dimensional analytical (MDA) queries*, while each data owner shares the data under LDP. An MDA query is a SQL query with aggregation (*e.g.*, COUNT, SUM, or AVG) on measure attributes and a predicate with multiple equality and range constraints on other attributes.

Demo overview. We propose and demonstrate a middleware solution **DPSAaS**, which enables **d**ifferentially **p**rivate data **s**haring and **a**nalytics **a**s cloud **s**ervices. Our vision with DPSAaS is to make differential privacy accessible to more "cloud users", who can be categorized into *data owners* and *data collectors* in DPSAaS.

A data owner holds one or multiple rows of multi-dimensional data that are sensitive but to be shared with a data collector. S/he

*Work done at Alibaba Group.
†Work done at Alibaba Group.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352085

	Age	Salary	State	OS	ActiveTime	Purchase
	D_1	D_2 (M_1)	D_3	D_4	D_5 (M_2)	D_6 (M_3)
t_1	30	50K	NY	Win	1.6h	\$120
t_2	60	80K	WA	iOS	1.2h	\$100
t_3	40	70K	NY	Win	1.0h	\$100
t_4	40	70K	NY	iOS	1.8h	\$100

Table 1: A relational table T with sensitive attributes

uses our *LDP data sharing service* in DPSAaS to encode the sensitive data into an LDP version (with the privacy budget s/he desires), which does not leak much information about each row (see Definition 1). The encoding algorithm (*e.g.*, via open source) and the encoding results are transparent to data owners. The LDP version of data can be submitted to the data collector for analytical tasks.

A data collector receives LDP version of data from a number of data owners (with the same data schema) and would like to conduct analytical study against them. Indeed, the normal query-processing engine gives meaningless output on the LDP-encoded data. Thus, the data collector uses our *LDP data analytics service* in DPSAaS, which is able to estimate answers to online MDA queries for the purpose of, *e.g.*, data exploration and trend analysis. An arbitrary number of MDA queries can be issued as the privacy is guaranteed before each data owner submits LDP-encoded data.

The audience for our demo can play both roles of data owners and data collectors. For those with little background in differential privacy, they can have better intuitions on why (L)DP is a reasonable privacy notation by observing the LDP-encoded data generated by our data sharing service from a data owner's perspective. For experts on private data analysis, they can check whether the query class supported and the accuracy satisfy their needs. For both, we would demonstrate the usability of DPSAaS services.

We will also highlight the design decisions of DPSAaS. Please refer to [10] for more related work and algorithmic details.

2. PRELIMINARIES

We introduce the data model, MDA queries supported by our DPSAaS, the privacy guarantee, and the algorithmic framework.

2.1 Data Model and MDA Queries

Suppose there are a set of *data owners*, each holding one or more tuples with the same set of *attributes*. Attributes are called *dimensions* when appearing in predicates of an analytical question, and are called *measures* when being aggregated to answer the question. We use t to denote a tuple, D to denote an attribute, M to denote a measure, and $t[D]$ or $t[M]$ to denote the attribute value in a tuple.

Conceptually, all tuples from all data owners can be collected by a *data collector* and form a *fact table* T. The data collector wants to ask analytical questions against T. We focus on the following class of *multi-dimensional analytical (MDA) queries*:

$$\texttt{SELECT}\ \mathsf{F}(M)\ \texttt{FROM}\ T\ \texttt{WHERE}\ \mathsf{C} \tag{2}$$

where the *aggregation* F is $\mathsf{COUNT}(*)$, $\mathsf{SUM}(M)$, or $\mathsf{AVG}(M)$; the *predicate* C consists of *point constraints* "$D_i = v_i$" for *categorical dimensions*, and *range constraints* "$D_i \in [l_i, r_i]$" for *ordinal dimensions* – we can support AND-OR expressions of constraints as predicates in this demo (*e.g.*, query Q_SUM in Section 1).

2.2 Local Differential Privacy (LDP)

All or some of the dimensions and measures are *sensitive*. Data owners *do not trust* the data collector and thus require formal privacy guarantees before they are willing to send their tuples. We adopt the *local model of differential privacy* (LDP) [3]. Under LDP, sensitive attributes in a tuple t are encoded with a randomized algorithm $\mathcal{A}$ by the data owner, and the output $\mathcal{A}(t)$ can be sent to the data collector. Intuitively, LDP guarantees that, no matter what $\mathcal{A}(t)$ is, it is approximately equally as likely to have come from t as any other t' differing from t in one or more sensitive attributes. Hence, as $\mathcal{A}(t)$, instead of t, is collected, t's information on sensitive attributes is protected (to some degree measured by the privacy budget ϵ), even if the data collector is malicious. Formally,

DEFINITION 1 (ϵ-LDP [3]). *Suppose $D_1, \ldots, D_d$ are sensitive attributes. A randomized algorithm $\mathcal{A}$ is ϵ-locally differentially private or ϵ-LDP, if for any pair of different tuples t and t', with $t[D_i] \neq t'[D_i]$ for at least one $i \in \{1, \ldots, d\}$, and any $O \subseteq \mathrm{Range}(\mathcal{A})$ (the domain of $\mathcal{A}$'s outputs),*

$$\mathbf{Pr}\left[\mathcal{A}(t) \in O\right] \leq e^{\epsilon} \cdot \mathbf{Pr}\left[\mathcal{A}(t') \in O\right].$$

2.3 Task and Algorithmic Framework

We demonstrate the task of *answering MDA queries under LDP* in DPSAaS. There are two components (services).

LDP data sharing service $\mathcal{A}$ (used by data owners). Each data owner runs an ϵ-LDP algorithm $\mathcal{A}$ on each of her/his tuples, t, and sends the output $\mathcal{A}(t)$, *i.e.*, the *LDP encoded tuple*, to the collector. The execution of $\mathcal{A}$ is independent of other tuples/data owners.

LDP data analytics service $\bar{\mathbf{P}}$ (used by data collectors). Suppose there are a total of n tuples from all data owners, forming a fact table $T = \{t_1, \ldots, t_n\}$. The data collector receives $\mathcal{A}(T) = \{\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n)\}$. An MDA query q, in the form of Equation (2), can be approximately answered on the *LDP encoded fact table* $\mathcal{A}(T)$ using an estimation algorithm $\bar{\mathbf{P}}$. Let $\bar{\mathbf{P}}(q, \mathcal{A}(T))$ be the estimate. *An arbitrary number* of queries can be issued, since LDP is preserved for each tuple t on the encoded tuple $\mathcal{A}(t)$ and the service $\bar{\mathbf{P}}(\cdot, \mathcal{A}(T))$ can be regarded as "post-processing" of them.

Error metric. Let $\mathbf{P}(q, T)$ denote the exact answer to an MDA query q against the fact table T. Let

$$\mathrm{Err}(\bar{\mathbf{P}}(q, \mathcal{A}(T))) = \mathbf{E}[(\bar{\mathbf{P}}(q, \mathcal{A}(T)) - \mathbf{P}(q, T))^2]$$

be the *expected (over randomness in $\mathcal{A}$) squared error*. The goal in [10] is to bound $\mathrm{Err}(\bar{\mathbf{P}}(q, \mathcal{A}(T)))$ for the supported MDA queries.

3. ARCHITECTURE OF DPSAaS

We first introduce some design decisions of DPSAaS, in order to smoothly connect DPSAaS to the data processing platform and analytics engines, and to facilitate efficient large-scale processing.

Overview of system design. DPSAaS is built as a middleware (Figure 1) on top of a(ny) data processing platform, *e.g.*, Spark.

The first question is where we implement and deploy our LDP algorithms, *i.e.*, LDP encoding algorithm $\mathcal{A}$ and estimation algorithm $\bar{\mathbf{P}}$. We have at least three alternatives: i) independent libraries, ii) inside of data processing engine, and iii) UDFs (*user-defined functions*). We eventually choose iii), *i.e.*, `LDP_Sharing_UDF` for $\mathcal{A}$ and `LDP_Analytics_UDAF` for $\bar{\mathbf{P}}$ in Figure 1, as it dominates the other two options in terms of both usability and efficiency. We only need to twist LDP algorithms to fit the UDF interfaces in the data processing platform. We will give more details in Section 3.1.

The second question is how these UDFs or UDAFs (*user-defined aggregation functions*) serve our users, *i.e.*, data owners and data collectors. Writing SQL statements with these UDFs and UDAFs for the purposes of data sharing and analytics, respectively, would be convenient for experts; however, there is a steep learning curve for non-expert data owners and collectors. Therefore, a component called "Sharing Query Generator" in the LDP data sharing service

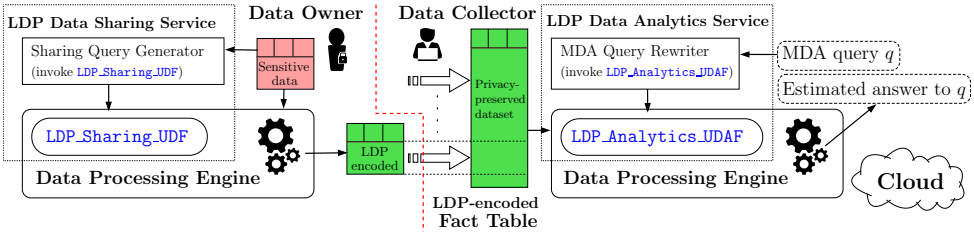

Figure 1: DPSAaS architecture

takes meta-information, *e.g.*, data schema and privacy budget, from the data owner, and generates a SQL statement automatically that invokes LDP_Sharing_UDF to encode each tuple under LDP. For a data collector, a component called "MDA Query Rewriter" in the LDP data analytics service takes an online MDA query (in the form of either a SQL statement or drag-and-drop in GUI), rewrites it to invoke LDP_Analytics_UDAF, and executes the rewritten query in the processing engine to estimate the answer to the original MDA query. We will give some examples in Section 3.2

We will discuss error bars associated with the query answers and how to interpret privacy budget in DPSAaS, in Sections 3.3-3.4.

3.1 Deploying LDP Algorithms via UDFs

We have considered three options to deploy LDP encoding algorithm $\mathcal{A}$ and estimation algorithm $\bar{\mathbf{P}}$, in our data sharing and data analytics services, respectively: i) implementing $\mathcal{A}$ and $\bar{\mathbf{P}}$ as independent libraries, ii) implementing them inside the data processing engine, and iii) implement them as *user-defined functions* (UDF) and *user-defined aggregation functions* (UDAF). We eventually choose iii), and implement LDP_Sharing_UDF for $\mathcal{A}$ and LDP_Analytics_UDAF for $\bar{\mathbf{P}}$. There are two major reasons.

The first reason is about usability and extensibility. If we implement $\mathcal{A}$ and $\bar{\mathbf{P}}$ as stand-alone libraries, we need to import these external libraries in the data processing platform and/or take care of data movement, which increases the complexity of our middleware. If we change the data processing engine to incorporate LDP, DPSAaS has to be tightly hooked up with a particular data platform, and is not easy to extended for others, not to mention the significant engineering efforts we have to spend for such deep integration.

The second one is about efficiency. More efforts have to be spent to make options i) and ii) able to process (including both sharing and analytics) large-scale datasets in a distributed way. The UDF and UDAF-based ways of deployment directly borrow the ability of distributed processing from the data platform itself.

Thus, we modify LDP algorithms to fit the following typical UDF and UDAF interfaces in the data processing platform.

```
class LDP_Sharing_UDF(BaseUDF):
  def evaluate(self, epsilon, data_tuple):
    # Implement the epsilon-LDP algorithm A
    # Output the LDP-encoded version of data_tuple
class LDP_Analytics_UDAF(BaseUDAF):
  def new_buffer(self):
    # Create a buffer to store partial sum of (3)
  def iterate(self, buf, ldp_tuple, query):
    # Process one LDP-encoded tuple as in RHS of (3)
    # Update partial sum of (3)
  def merge(self, buf, pbuf):
    # Merge intermediate results (partial sums)
  def terminate(self, buf):
    # Final output (estimated answer)
```

It is straightforward to implement the LDP encoding algorithm $\mathcal{A}$ in the method `evaluate` as the input to $\mathcal{A}$ is also one tuple.

For the estimation algorithm $\bar{\mathbf{P}}(q, \mathcal{A}(T))$, which takes the query q and the LDP-encoded fact table $\mathcal{A}(T)$ as inputs, we need to decompose it as follows to fit interface of LDP_Analytics_UDAF:

$$\bar{\mathbf{P}}(q, \mathcal{A}(T)) = \sum_{t_{\text{ldp}} \in \mathcal{A}(T)} \bar{\mathbf{P}}(q, t_{\text{ldp}}). \tag{3}$$

Many LDP estimation algorithms have such decomposability, including ours [10]. $\bar{\mathbf{P}}(q, t_{\text{ldp}})$ is implemented in `iterate` with partial sums of (3) being computed in parallel. Partial sums are merged in `merge`, and `terminate` gives the final estimated answer to q.

3.2 Query Generating and Rewriting

Per our early discussion, following is an example generated by "Sharing Query Generator" for sharing sensitive data in Table 1 (the first three attributes are sensitive and are encoded under 2-LDP):

```
INSERT OVERWRITE TABLE ldp_T
SELECT LDP_Sharing_UDF(2.0, Age, Salary, State) as
       ldp_tuple, OS, ActiveTime, Purchase FROM T;
```

Now, "MDA Query Rewriter" rewrites Q_SUM, (1) in Section 1, to invoke the analytics UDAF and estimate its answer from `ldp_T`:

```
SELECT LDP_Analytics_UDAF(ldp_tuple, Purchase,
       Q_SUM_Str) FROM ldp_T;
```

where `Q_SUM_Str` is a string representation of Q_SUM (input q to $\bar{\mathbf{P}}$) and will be parsed inside LDP_Analytics_UDAF; `ldp_tuple` are the LDP version of attributes collected from data owners.

$\mathcal{A}(t)$ and $\bar{\mathbf{P}}(q, t_{\text{ldp}})$ as UDF/UDAF are efficient, with costs linear in the size of a tuple [10]. Via query generating and rewriting, the efficiency of the two services in DPSAaS is further boosted by the ability of distributed processing of data platforms, *e.g.*, Spark.

3.3 Estimating Error Bars

Recall that our LDP data analytics service gives an "estimated answer" to an MDA query. We have theoretical bounds [10] of the error metric introduced in Section 2.3. However, the big-O notations in [10] hide some constants in the errors. When it comes to showing error bars on the estimated answers to users, confidence intervals or variances could be more intuitive and accurate.

Specifically, given a query and its estimated answer, we highlight an error bar which means that the true answer lies within this range with probability over, *e.g.*, 90%. This is easy for COUNTqueries, for which error bars can be calculated from variance or (α, β)-accuracy; for SUM queries, we can rely on our variance analysis to derive approximate confidence intervals; and for an AVG query, whose answer is derived by dividing SUM with COUNT, we can divide the error bars of the two to obtain a confidence interval.

3.4 Privacy Budget and Implications

DPSAaS guarantees ϵ-LDP for each tuple during the data sharing service against the data collector. There are two important notes about this guarantee. First, it is dangerous to run the ϵ-LDP algorithm $\mathcal{A}$ on the same tuple multiple times; no matter which output(s) of these runs is (are) submitted to the data collector, the privacy guarantee would be weakened – for example, if outputs of k

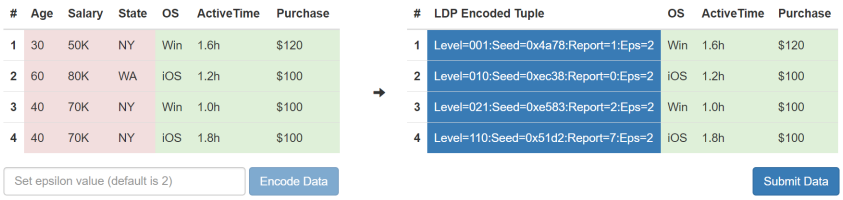

Figure 2: Interface for data owners (`T` → `ldp_T`)

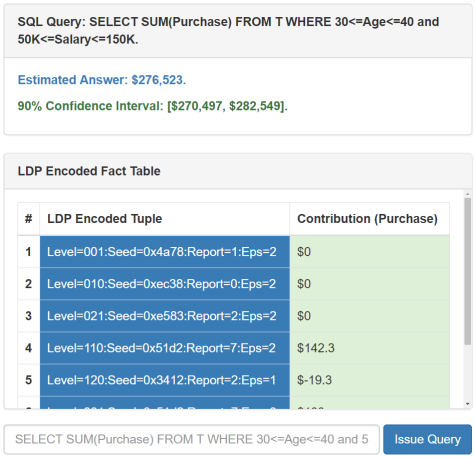

Figure 3: Interface for data collectors/analysts

runs of ϵ-LDP $\mathcal{A}$ are released, we can only guarantee $k\epsilon$-LDP overall. Secondly, if there are multiple tuples about the same individual in the fact table, *e.g.*, a tuple is about an individual's daily activity (one tuple per day), DPSAaS guarantees ϵ-LDP per tuple, but not per individual (not protecting one's long-term behavior).

Meanwhile, an arbitrary number of MDA queries can be issued without using up the privacy budget, because LDP is preserved for each tuple in the data sharing service, and the analytics service can be regarded as "post-processing" of the LDP encoded fact table.

4. DEMO OVERVIEW

We demonstrate DPSAaS for two different scenarios. Namely, *data sharing*, where the audience role plays as a data owner who uses the LDP data sharing service in DPSAaS to share sensitive data under LDP; and *data analytics*, where the audience pretends to be a data analyst who has collected LDP encoded data and wish to analyze them by issuing MDA queries via our LDP data analytics service. We use Spark as the underlying data processing platform here (DPSAaS can be easily plugged into other platforms, too).

4.1 Sharing Sensitive Data

The goals of our demo for this scenario are to (a) illustrate how easy it is to use our service to enforce LDP in the data to be collected/shared; and (b) show the LDP-encoded data to their owners to give intuitions on why (L)DP is a reasonable privacy notation.

Figure 2 shows a simple GUI used by the each data owner to share her/his sensitive data. Data owners can browse their original data. After specifying the privacy budget ϵ (*different owners can specify different values of* ϵ) and click the "Encode Data" button, the ϵ-LDP encoding algorithm $\mathcal{A}$ is run on each tuple to enforce LDP. The data owner can view the encoded results on the right side, and choose to "Submit" the LDP encoded data to the data collector.

Figure 2 gives some samples for "LDP encoded tuples" based on our techniques in [10]. Note that in general, all attributes can be sensitive in DPSAaS. In this example, we assume that Age, Salary, and State are sensitive. Even when two tuples have the same values on all the three attributes (*e.g.*, #3 and #4), the corresponding LDP encoded versions could be different (due to randomness in algorithm $\mathcal{A}$), which by itself is a good property for privacy protection.

Attendees will be able to choose different datasets, and try different settings such as different values of ϵ and different sets of sensitive attributes to see how they affect the query results.

4.2 Private Data Analytics

After receiving LDP encoded tuples from data owners, the data collector/analyst can issue MDA queries. Here, our goals are to (a) illustrate how to issue MDA queries against LDP encoded data; (b) give intuitions on how to process them; and (c) show how privacy budget ϵ affects errors in the estimated answers.

Figure 3 shows a GUI for data analysts. A data analyst can browse all the LDP encoded tuples received, which do not leak any per-tuple information about sensitive attributes. S/he can write an MDA query as a SQL statement (against the original data), and click the "Issue Query" button; DPSAaS will then rewrite this query to invoke "LDP_Analytics_UDAF", execute it in the processing engine, and display the estimated answer. A drag-and-drop query builder and visualization of estimated answers can also be supported for analysts who are not familiar with SQL.

During the processing of the MDA query, the column "Contribution" in the "Fact Table" is updated. It shows the contribution of each LDP encoded tuple towards the final estimated answer on the measure (Purchase in this example). More precisely, it is the RHS of (3), and is query-dependent. It can be seen that even when two tuples have the same values on sensitive attributes and measures (*e.g.*, #3 and #4), they may contribute differently to the estimation, which, again, is a good property for privacy protection.

We will have the exact answer to the same query on the side for attendees to compare and find out how the error is affected by ϵ. The attendees are allowed to issue as many different queries as possible without further concerns about privacy.

5. REFERENCES

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

[2] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In *NIPS*, pages 3574–3583, 2017.

[3] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *FOCS*, pages 429–438, 2013.

[4] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[5] Ú. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *CCS*, pages 1054–1067, 2014.

[6] N. M. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for SQL queries. *PVLDB*, 11(5):526–539, 2018.

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

[8] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, pages 19–30, 2009.

[9] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis. *PVLDB*, 7(8):637–648, 2014.

[10] 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*, pages 159–176, 2019.