Continuous Release of Data Streams under both Centralized and Local Differential Privacy

Tianhao Wang*
Carnegie Mellon University & University of Virginia

Joann Qiongna Chen
University of California, Irvine

Zhikun Zhang
CISPA

Dong Su
Alibaba Inc.

Yueqiang Cheng
NIO Security Research

Zhou Li
University of California, Irvine

Ninghui Li
Purdue University

Somesh Jha
University of Wisconsin, Madison

ABSTRACT

We study the problem of publishing a stream of real-valued data satisfying differential privacy (DP). One major challenge is that the maximal possible value in the stream can be quite large, leading to enormous DP noise and bad utility. To reduce the maximal value and noise, one way is to estimate a threshold so that values above it can be truncated. The intuition is that, in many scenarios, only a few values are large; thus truncation does not change the original data much. We develop such a method that finds a suitable threshold with DP. Given the threshold, we then propose an online hierarchical method and several post-processing techniques.

Building on these ideas, we formalize the steps in a framework for the private publishing of streaming data. Our framework consists of three components: a threshold optimizer that privately estimates the threshold, a perturber that adds calibrated noise to the stream, and a smoother that improves the result using post-processing. Within our framework, we also design an algorithm satisfying the more stringent DP setting called local DP. Using four real-world datasets, we demonstrate that our mechanism outperforms the state-of-the-art by a factor of $6-10$ orders of magnitude in terms of utility (measured by the mean squared error of the typical scenario of answering a random range query).

CCS CONCEPTS

• **Information systems** → **Data streams**; • **Security and privacy** → **Privacy-preserving protocols**.

KEYWORDS

Differential Privacy; Local Differential Privacy; Continuous Observation; Data Stream

ACM Reference Format:
Tianhao Wang, Joann Qiongna Chen, Zhikun Zhang, Dong Su, Yueqiang Cheng, Zhou Li, Ninghui Li, and Somesh Jha. 2021. Continuous Release of Data Streams under both Centralized and Local Differential Privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3460120.3484750

*Tianhao did most of the work while at Purdue University.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484750

1 INTRODUCTION

Continuous observation over data streams has been utilized in several real-world applications. For example, security companies continuously analyze network traffic to detect abnormal Internet behaviors [9]. However, analyzing and releasing streams raise privacy concerns when these data contain sensitive individual information. Directly publishing raw statistics may reveal individual users' private information. For instance, electricity usage data from smart meters can reveal whether a user is at home or even what household appliances are used at some specific time [34].

A promising technique for releasing private statistics is differential privacy (DP) [17], which has become the gold standard in the privacy-research community. Informally, any algorithm satisfying DP has the property that its output distribution on a given database is close to the output distribution on a similar dataset where any single record is replaced. The closeness is quantified by a parameter ϵ, where a smaller ϵ offers a better privacy guarantee.

To publish streams with DP, a widely accepted approach is to use the hierarchical structure [8, 18]. The idea is to partition the time series into multiple granularities and then add noise to the stream to satisfy DP. Because of the additive noise, it is impossible to accurately publish any single value in the stream. Thus our goal is to *accurately estimate the sum of values over any range of time.* One challenge is that to satisfy DP, the magnitude of the noise should be proportional to the upper bound of the data, which is typically large. Perrier et al. [36] (termed PAK in this paper) observed that data in the stream is often concentrated below a value much smaller than the upper bound. To exploit this insight, a technique called contribution limitation is commonly-used [13, 30, 32, 50, 54]. It truncates the data using a specified threshold θ (i.e., values larger than θ are replaced by θ). The rationale is to reduce the noise (now the noise is proportional only to θ) while preserving utility. To find such a threshold while maintaining DP, PAK developed a method based on smooth sensitivity [35]. The result can then be applied to the hierarchical algorithm to publish streams with improved utility.

We find three key limitations in existing work of PAK's. First, it tries to privately find the 99.5-th percentile to serve as the threshold θ. Unfortunately, using the 99.5-th percentile (or any other fixed percentile) is unlikely to work across all settings of ϵ values and

data distributions. Second, in order to get an analytical upper bound of the error caused by truncation (this error is also called bias), the authors further increase the estimated 99.5-th percentile by first adding a positive term, and then multiplying a coefficient greater than 1. As a result, the chosen θ is often unnecessarily large. When ϵ is small (e.g., $\epsilon \leq 0.1$), the value of θ is usually larger than the maximal possible value, running against the original purpose of choosing the threshold. Third, the method directly utilizes a basic hierarchical approach to output the stream, and does not fully take advantage of post-processing optimizations. As a consequence, the accuracy of the output is far from ideal, and the results are worse when answering range queries with small selectivity.

In this paper, we propose a new approach by addressing the above-mentioned three limitations. Instead of using a fixed percentile, we design a data-dependent method to find the threshold θ that considers the overall data distribution. Our goal is to minimize the overall error due to bias and DP noise simultaneously. Given θ, we then propose a new hierarchical algorithm to obtain accurate results. One major contribution is a novel online algorithm to enforce consistency over the noisy estimates (i.e., to make sure the number on any node equal the sum of its children's, which is violated if independently sampled noise is added to the hierarchy) on the hierarchy to provide better utility. While there exists consistency methods that work on the noisy hierarchies, our observation is that we can pre-compute all the noise, and then make the noise consistent first. As the true values are naturally consistent, we can then add the consistent noise to the true values in an online manner and thus achieve an online consistency algorithm. We prove the algorithm achieves minimum squared error and also satisfies DP. Another contribution is that we further extend the algorithm to prune the lower-level nodes based on an optimization criterion, based on the observation that the estimates in the lower levels of the hierarchy tend to be overwhelmed by the noise, leading to a low signal-noise ratio. Our new hierarchical algorithm is also able to handle infinite streams.

Next, we generalize the above-mentioned algorithms into a new framework for streaming data publication. It consists of three components: a *Threshold optimizer*, a *Perturber*, and a *Smoother*. The threshold optimizer consumes a portion of the input stream, and finds a threshold θ. It then truncates all incoming values by θ and sends them to the perturber. The perturber adds noise to each incoming element of the stream, and releases noisy counts to the smoother. Finally, the smoother performs further post-processing on the noisy counts and outputs the final stream. Together with the new algorithms described above, we call our solution *ToPS*.

Finally, based on the framework of ToPS, we design an algorithm to output streams while satisfying local DP (LDP), which protects privacy under a stronger adversary model than DP. We call the resulting method ToPL. Under LDP, only the users know the true values and thus removes the dependence on the trusted central server. In ToPL, we use state-of-the-art LDP mechanisms for the Threshold optimizer and the Perturber. While the design of ToPL relies on the findings in ToPS, we also adapt existing LDP mechanisms to our setting to get better performance.

We implemented both ToPS and ToPL, and evaluated them using four real-world datasets, including anonymized DNS queries, taxi trip records, click streams, and merchant transactions. We use the Mean Squared Error (MSE) over random range queries as the metric of performance evaluation. The experimental results demonstrate that our ToPS significantly outperforms the previous state-of-the-art algorithms. More specifically, the most significant improvement comes from our new technique to finding θ. It contributes an improvement of $4-8$ orders of magnitude over PAK. Even given the same reasonable θ, ToPS can answer range queries $100\times$ more accurately than PAK. Putting the two together, ToPS improves over PAK by $6-10$ orders of magnitude in terms of MSE.

Contributions. To summarize, the main contributions of this paper are threefold:

- We design ToPS for releasing real-time data streams under differential privacy. Its contributions include an EM-based algorithm to find the threshold, an online consistency algorithm, the use of a smoother to reduce the noise, and the ability to handle infinite streams.
- We extend ToPS to solve the problem in the more stringent setting of LDP and propose a new algorithm called ToPL.
- We evaluate ToPS and ToPL using several real-world datasets. The experimental results indicate that both can output streams accurately in their settings. Moreover, ToPS outperforms the previous state-of-the-art algorithms by a factor of $6-10$ orders. Our code is open sourced at https://github.com/dp-cont/dp-cont.

Roadmap. In Section 2, we present the problem definition and the background of DP and LDP. We present the existing solutions and our proposed method in Section 3 and 4. Experimental results are presented in Section 5. Finally, we discuss related work in Section Section 6 and provide concluding remarks in Section Section 7.

2 PROBLEM DEFINITION AND PRELIMINARIES

We consider the setting of publishing a stream of real values under differential privacy (DP). The length of the stream could be unbounded. Due to the constraint of DP, it is unrealistic to make sure every single reading of the stream is accurate, so the goal is to ensure the aggregated estimates are accurate.

2.1 Formal Problem Definition

There is a sequence of readings $V = \langle v_1, v_2, \ldots \rangle$, each being a real number in the range of $[0, B]$. We publish a private sequence $\tilde{V}$ of the same size as V while satisfying DP, with the goal of accurately answering range queries. Range query is an important tool for understanding the overall trend of the stream. Specifically, a range query $V(i, j)$ is defined as the sum of the stream from index i to j, i.e., $V(i, j) = \sum_{k=i}^{j} v_k$. We want a mechanism that achieves a low expected squared error of any randomly sampled range queries, i.e.,

$$\mathbb{E}\left[\left(\tilde{V}(i, j) - V(i, j)\right)^2\right]. \tag{1}$$

2.2 Differential Privacy

We follow the setting of PAK [36] and adopt the notion of *event-level* DP [18], which protects the privacy of any value in the stream.

Definition 2.1. (Event-level (ϵ, δ)-DP) An algorithm $\mathbf{A}(\cdot)$ satisfies (ϵ, δ)-differential privacy ((ϵ, δ)-DP), if and only if for any two neighboring sequences V and V' and for any possible output set O,

$$\Pr[\mathbf{A}(V) \in O] \leq e^{\epsilon} \Pr\left[\mathbf{A}(V') \in O\right] + \delta,$$

where two sequences $V = \langle v_1, v_2, \ldots\rangle$ and $V' = \langle v'_1, v'_2, \ldots\rangle$ are neighbors, denoted by $V \simeq V'$, when $v_i = v'_i$ for all i except one index.

For brevity, we use (ϵ, δ)-DP to denote Definition 2.1. When $\delta = 0$, which is the case we consider in this paper, we omit the δ part and write ϵ-DP instead of $(\epsilon, 0)$-DP.

Justification of Event-Level DP. Although event-level DP only protects one value, it is a suitable guarantee in many cases. For example, individuals might be happy to disclose their routine trip to work while unwilling to share the occasional detour. Note that the data model is general and V can also come from multiple users. For example, V consists of the customers' expenditure from a grocery store, and we want to protect some unusual transaction. Moreover, our model is a generalization of the basic model where every value is binary [8, 21], and can be used in building private algorithms with trusted hardware [7].

Extension to Event-Level LDP. We also work in the local version of DP [28]. Compared to the centralized setting, local DP offers a stronger trust model, because each value is reported to the server in a perturbed form. Privacy is protected even if the server is malicious. For each value v in the stream of V, we have the following guarantee:

Definition 2.2 ((ϵ, δ)-LDP). An algorithm $\mathbf{A}(\cdot)$ satisfies (ϵ, δ)-local differential privacy ((ϵ, δ)-LDP), if and only if for any pair of input values v, v', and any set O of possible outputs of $\mathbf{A}$, we have

$$\Pr[\mathbf{A}(v) \in O] \leq e^{\epsilon} \Pr\left[\mathbf{A}(v') \in O\right] + \delta.$$

Typically, $\delta = 0$ in LDP [33, 41, 45, 49] (one reason is that many LDP protocols are built on randomized response [49], which ensures $\delta = 0$). Thus we simplify the notation and call it ϵ-LDP. The notion of LDP differs from DP in that each user perturbs the data before sending it out and thus do not need to trust the server under LDP.

2.3 Mechanisms of Differential Privacy

We first review primitives proposed for satisfying DP. We defer the descriptions of LDP primitives to Appendix E as our LDP method mostly uses the LDP primitives as blackboxes.

Laplace Mechanism. The Laplace mechanism computes a function f on the input V in a differentially private way, by adding to $f(V)$ a random noise. The magnitude of the noise depends on GS_f, the *global sensitivity* or the L_1 sensitivity of f, defined as,

$$\mathrm{GS}_f = \max_{V \simeq V'} ||f(V) - f(V')||_1.$$

When f outputs a single element, such a mechanism $\mathbf{A}$ is given below:

$$\mathbf{A}_f(V) \quad = f(V) + \mathsf{Lap}\left(\frac{\mathrm{GS}_f}{\epsilon}\right).$$

In the definition above, $\mathsf{Lap}(\beta)$ denotes a random variable sampled from the Laplace distribution with scale parameter β such that $\Pr[\mathsf{Lap}(\beta) = x] = \frac{1}{2\beta} e^{-|x|/\beta}$, and it has a variance of $2\beta^2$. When f outputs a vector, $\mathbf{A}$ adds independent samples of $\mathsf{Lap}\left(\frac{\mathrm{GS}_f}{\epsilon}\right)$ to each element of the vector.

Noisy Max Mechanism. The Noisy Max mechanism (NM) [20] takes a collection of queries, computes a noisy answer to each query, and returns the index of the query with the largest noisy answer.

More specifically, given a list of queries $q_1, q_2, \ldots$, where each q_i takes the data V as input and outputs a real-numbered result, the mechanism computes $q_i(V)$, samples a fresh Laplace noise $\mathsf{Lap}\left(\frac{2\mathrm{GS}_q}{\epsilon}\right)$ and adds it to the query result, i.e.,

$$\tilde{q}_i(V) = q_i(V) + \mathsf{Lap}\left(\frac{2\mathrm{GS}_q}{\epsilon}\right), \tag{2}$$

and returns the index $j = \arg\max_i \tilde{q}_i(V)$. Here GS_q is the global sensitivity of queries and is defined as:

$$\mathrm{GS}_q = \max_i \max_{V \simeq V'} |q_i(V) - q_i(V')|.$$

Dwork and Roth prove this satisfies ϵ-DP [20] (recently Ding et al. [14] proved using exponential noise also satisfy DP). Moreover, if the queries satisfy the monotonic condition, meaning that when the input dataset is changed from V to V', the query results change in the same direction, i.e., for any neighboring V and V'

$$\left(\exists_i\, q_i(V) < q_i(V')\right) \implies \left(\forall_{i'}\, q_{i'}(V) \leq q_{i'}(V')\right).$$

Then one can remove the factor of 2 in the Laplace noise. This improves the accuracy of the result.

2.4 Composition Properties

The following composition properties hold for both DP and LDP algorithms, each commonly used for building complex differentially private algorithms from simpler subroutines.

Sequential Composition. Combining multiple subroutines that satisfy DP for $\epsilon_1, \cdots, \epsilon_k$ results in a mechanism that satisfies ϵ-DP for $\epsilon = \sum_i \epsilon_i$.

Parallel Composition. Given k algorithms working on disjoint subsets of the dataset, each satisfying DP for $\epsilon_1, \cdots, \epsilon_k$, the result satisfies ϵ-DP for $\epsilon = \max_i \epsilon_i$.

Post-processing. Given an ϵ-DP algorithm $\mathbf{A}$, releasing $g(\mathbf{A}(V))$ for any g still satisfies ϵ-DP. That is, post-processing an output of a differentially private algorithm does not incur any additional loss of privacy.

3 DIFFERENTIALLY PRIVATE STREAMS

For privately releasing streams and supporting range queries over the private stream, the most straightforward way is to add independent noise generated through the Laplace distribution. However, this results in a cumulative error (following the tradition, we use absolute error here, which measures the difference from the true sum) of $O(\sqrt{n})$ after n observations.

The Hierarchy Approach. To get rid of the dependency on $\sqrt{n}$, the hierarchical method was proposed [8, 18]. Given a stream of length n, the algorithm first constructs a tree: the leaves are labeled $\{1\}, \{2\}, \ldots, \{n\}$ and the label of each parent node is the union of labels from its child nodes. Given $h = \log n$ layers, the method adds

Laplace noise with ϵ/h in each layer. To obtain the noisy count $\tilde{V}(i,j)$, we find at most $\log n$ nodes in the hierarchy, whose labels are disjoint and their union equals $[i,j]$. Given that the noise added to each node is $O(\log n)$, this method has an error of $O(\log^{1.5} n)$.

In the online setting, where the stream data come one-by-one, we want to release every node in the hierarchy promptly. To do so, at any time index t, we publish all nodes that contain t as the largest number in their labels.

3.1 Existing Work: PAK

To satisfy DP in the hierarchy method, one needs to add noise proportional to B, the maximal possible value in the stream, and B can be quite large in many cases. Perrier et al. [36] (we call it by the authors' initials, PAK, for short) observed that in practice, most of the values are concentrated below a threshold much smaller than B (e.g., the largest possible purchase price of supermarket transactions is much larger than what an ordinary customer usually spends), and proposed a method to find such a threshold and truncate data points below it to reduce the scale of the injected Laplace noises. In particular, the first m values are used to estimate the threshold θ with differential privacy. After obtaining θ, the following values in the stream are truncated to be no larger than θ. Reducing the upper bound from B to θ reduces the DP noise (via reducing sensitivity). The hierarchical method is used for estimating the stream statistics with the remaining $n-m$ values (PAK assumes there are n observations).

Finding the Threshold. To obtain θ, PAK proposed a specially designed algorithm based on Smooth Sensitivity (SS) [35] to get the p-quantile (or p-percentile) as θ, i.e., $p\%$ of the values are smaller than θ. SS was used to compute the median with DP in the original work [35], and SS can also be easily extended to privately release the p-quantile. PAK proved that the result of SS is unbiased, but they further wanted to make sure the result is always larger than the real p-quantile. This is because if the estimated percentile is smaller than the real one, the truncation in the next phase will introduce greater bias. Thus PAK modified the original SS method to guarantee that the result is unlikely to be smaller than the real p-quantile. As the details of the method are not directly used in the rest of the paper, we defer the details of both SS and the algorithm itself to Appendix A and B.

There are two drawbacks of this method. First, it requires a p value to be available beforehand. But a good choice of p actually depends on the dataset, ϵ, and m. PAK simply uses $p = 99.5$. As shown in our experiment in Section 5.4, $p = 99.5$ does not perform well in every scenario. Second, to ensure that θ is no smaller than the real p-quantile, PAK introduces a positive bias to θ.

3.2 Overview of Our Approach

The design of PAK was guided by asymptotic analysis. Unfortunately, for the parameters that are likely to occur in practice, the methods and parameters chosen by asymptotic analysis can be far from optimal, as such analysis ignores important constant factors. Instead, we use concrete analysis to guide the choice of methods and parameters.

In this section, we first deal with the threshold selection problem using the Noisy Max mechanism (NM, introduced in Section 2.3), which satisfies DP with $\delta = 0$. Empirical experiments show its superiority especially in small ϵ scenarios (which means compared to PAK, we can achieve the same performance with better privacy guarantees). We then introduce multiple improvements for the hierarchical methods including an online consistency method, and a method to reduce the noise in the lower levels of the hierarchy. We integrate all the components in a general framework, *ToPS*, which consists of *a Threshold optimizer, a Perturber, and a Smoother*:

- *Threshold optimizer*: The threshold optimizer uses a small portion of the input stream to find a threshold θ for optimizing the errors due to noise and bias. It then truncates any incoming values by θ and releases them to the perturber.
- *Perturber*: The perturber adds noise to the truncated stream, and releases noisy counts to the smoother.
- *Smoother*: The smoother performs further post-processing on the noisy counts and outputs the final stream.

While design of ToPS is inspired by PAK, we include unique design choices to handle the problem. In particular, PAK has two phases, the threshold finder and the hierarchical method. We improve both phases. Note that as we use NM [20], our algorithm satisfies ϵ-DP while PAK satisfies (ϵ,δ)-DP. Moreover, we introduce a smoother that further improves accuracy. The fundamental reason that our method works better is that we design our method focusing on a good empirical performance rather than theoretical bounds.

3.3 Threshold Optimizer

Different from PAK [36] that focuses on bias when choosing the threshold θ, our approach is to consider both bias and variance (due to DP noise). As bias and variance go in opposite ways (i.e., when θ is large, bias will be small, but variance will go large, vice versa), there will be an optimal θ that minimizes the overall error. Note that the overall error depends on the whole distribution of the data, which might be too much information to accurately estimate with DP. To handle this issue, we choose to use the Noisy Max mechanism (NM) [20], which looks into the data privately and outputs only a succinct information of θ. In what follows, we first examine the error.

A Basic NM Query Definition. We first consider the expected squared error of estimating a single value v. Assuming that $\tilde{v}$ is the estimation of v, it is well known that the expected squared error is the summation of variance and the squared bias of $\tilde{v}$:

$$\mathbb{E}\left[(\tilde{v}-v)^2\right] = \mathrm{Var}[\tilde{v}] + \mathrm{Bias}[\tilde{v}]^2 . \tag{3}$$

Note that $\mathrm{Bias}[\tilde{v}]$ equals $\mathbb{E}[\tilde{v}]-v$. Given a threshold θ and privacy budget ϵ, $\mathrm{Bias}[\tilde{v}] = \max(v-\theta,0)$ and $\mathrm{Var}[\tilde{v}] = \frac{2\theta^2}{\epsilon^2}$ (because we add Laplace noise with parameter θ/ϵ).

Since we are using the hierarchical method to publish streams for answering range queries, we use the error estimations of the hierarchical method to instantiate Equation 3. Qardaji et al. [37] show that there are approximately $(b-1)\log_b(r)$ nodes to be estimated given any random query of range smaller than r, where b is the fan-out factor of the hierarchical method; and the variance of each node is $\log_b^2(r)\frac{2\theta^2}{\epsilon^2}$. For bias, within range limitation r, a random query will cover around $\frac{r}{3}$ leaf nodes on average [37] (We assume a random query can take any range in $[1,r]$. Thus there

are $r(r+1)/2$ possible range queries. Among them, for any range of length $j \in [1, r]$, there are $r - j + 1$ such ranges. The expected length of a random query is $\frac{\sum_{j=1}^{r} j(r-j+1)}{r(r+1)/2} = \frac{r+2}{3} \approx \frac{r}{3}$). Denote f_t as the frequency of value t, the combined error of the hierarchical method for answering random range queries would be:

$$(b-1)\log_b^3(r)\frac{2\theta^2}{\epsilon^2} + \left(\frac{r}{3}\sum_{\theta<t<B} f_t(t-\theta)\right)^2. \quad (4)$$

The Final NM Query Definition. For NM to be effective, the queries should have low sensitivity, meaning that changing one value perturbs the queries by a tiny amount. However, if we directly use Equation 4 as the queries, the sensitivity is large: a change of value from 0 to B will result in the increase of Equation 4 by $(B-\theta)^2/9$. Thus we choose to approximate the mean squared error by defining the queries in the following ways.

Denote m as the number of values to be used in NM, and m_θ as the number of values that are smaller than θ from these m values:

$$m_\theta = |\{i \mid v_i \le \theta, i \in [m]\}|, \quad (5)$$

where $[x] = \{1, 2, \ldots, x\}$ and $|X|$ denotes the cardinality of set X. The first approximation method we use is to replace the variance and squared bias with their squared roots (standard deviation and bias). Second, we use $c \cdot m_\theta/m$ (c is a constant to be discussed later) to approximate $\sum_{\theta<t<B} f_t(t-\theta)$. Third, we multiply both the standard deviation and bias errors by $-\frac{3m}{c \cdot r}$ to ensure the sensitivity is 1, and the query result of the target is the highest. Thus we have:

$$\begin{aligned} q_\theta(V) &= -\frac{3m}{c \cdot r}\sqrt{(b-1)\log_b^3(r)\frac{2\theta^2}{\epsilon^2}} - m_\theta \\ &= -\frac{3m\theta}{cr\epsilon}\sqrt{2(b-1)\log_b^3(r)} - m_\theta. \end{aligned} \quad (6)$$

The first term is a constant depending on θ but independent of the private data, while the second term has a sensitivity of 1.

Running NM. To run NM, the set of possible θ values considered in the queries $q_\theta(V)$ should be a discrete set that covers the range $[0, B]$. The granularity of the set is important. If it is too coarse-grained (e.g., $\theta \in \{0, B/2, B\}$), the method is inaccurate, because the desired value might be far from any possible output. On the other hand, if it is too fine-grained, the NM algorithm will run slowly, but it does not influence the accuracy. In the experiment, we use all integers in the range of $[B] = \{1, 2, \ldots, B\}$ as the possible set of θ.

One unexplained parameter in Equation 6 is c. There are two factor that contributes to c: (1) Using m_θ/m to approximate the bias term $\sum_{\theta<t<B} f_t(t-\theta)$ leads to underestimation. (2) As we will describe later in Section 3.4 and 3.5, the actual squared error will be further reduced by our newly proposed method. While c intends to be a rough estimation of the underestimation, it does not need to be chosen based on one particular dataset. One can run experiments with a public dataset of similar nature under different parameters, the best level of error that can be achieved is usually a good indicator of c. When a public dataset is unavailable, one can generate a synthetic dataset under some correlation assumption and run experiments. In experiments conducted for this paper, we choose $c = 60$, and use it for all datasets and settings.

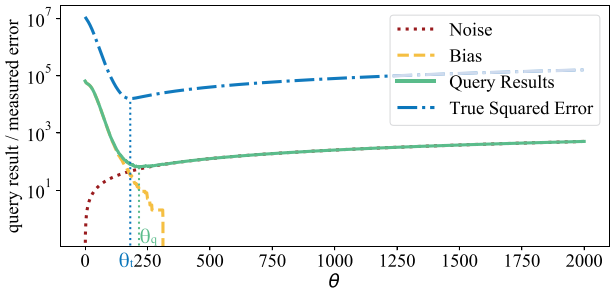


Figure 1: Empirical comparison of approximated query results (Equation 6) and the true squared errors and their minimum points θ_q and θ_t on a real-world datasets (DNS). We use $\epsilon = 0.1, m = 2^{16}, r = 2^{20}$. The x-axis is the possible value of θ, and the y-axis is the query result or the measured error.

Verify the Approximation. Figure 1 illustrates the distribution of Equation 6 and measured errors on a dataset that is used in the experiments in Section 5. The dataset is a network streaming dataset, called DNS. We use $\epsilon = 0.1, m = 2^{16}, r = 2^{20}$, which are the same as those parameters used in experiments. From Figure 1, we can see that the distributions between the truly measured errors (Equation 4) and the corresponding Equation 6 on two datasets are very close. The figure also illustrates the bias and variance factors of Equation 6. The two factors grow in opposite directions which makes a global minimum where the target θ lies. In addition, we also show that the threshold θ_q that minimizes our queries Equation 6 is close to the target threshold θ_t which minimizes the real measured errors (Equation 4). Therefore, the above empirical evaluation results show the capability of the threshold optimizer in finding accurate θ values.

3.4 Perturber

The perturber inherits the hierarchical idea [8, 18] (also described in Section 3.1). In this section, we start from the binary hierarchy used in PAK and put together three improvements to it to obtain a solution that is practical across a wide range of datasets.

1. Better Fan-out. According to Qardaji et al. [37], using a fan-out $b = 16$ instead of 2 in the hierarchy can give better utility. The result of optimal fan-out $b = 16$ is derived by analyzing the accuracy (variance) of answering range queries. In particular, we assume the range query is random and all layers in the hierarchy receive the same amount of privacy budget. We then measure the expected accuracy (measured by variance) of answering the range query. The optimal value $b = 16$ is obtained by minimizing the variance. It does not change on different datasets because the analysis is data-independent. We thus use fan-out $b = 16$ by default.

2. Handling Infinite Streams. PAK requires a fixed length n a priori in order to build the hierarchy. As a result, their algorithm stops after n observations. In order to support infinite streams, Chan et al. [8] proposed to have an infinitely high hierarchy, and each layer receives a privacy budget inversely proportional to the height of the layer. For example, the bottom layer receives 0.9ϵ,

then its parent layer receives $0.9^2\epsilon$, and so on. While this ensures the overall privacy budget will never exceed ϵ, the higher layers are essentially receiving a tiny amount of privacy budget.

In this paper, we note that most of the queries focus on limited ranges, and propose to have an upper bound on the query range denoted by r and then split ϵ equally to the $h = \lceil \log_b r \rceil$ layers. The value of r stands for a limit below which most queries' ranges fall, and is determined externally (e.g., if we receive one value per minute, then it is unlikely that a query spans over a year). For each chunk of r observations, we output a height-h hierarchy. We note that each hierarchy handles a disjoint sub-stream of observations and thus this extension does not consume additional privacy budget because of the parallel composition property of differential privacy. In the evaluation, we choose $r = 2^{20}$.

3. Online Consistency. Given a noisy hierarchy, Hay et al. [26] proposed an efficient algorithm for enforcing consistency among the values in it. By enforcing consistency, the accuracy of the hierarchy can be improved. Note that this is a post-processing step and does not consume any privacy budget. Unfortunately, the algorithm is *off-line* and requires the whole hierarchy data to be available. Here we propose an *online* version of the enforce-consistency algorithm, so that we can output the noisy streams promptly.

Our method is built on the work of Hay et al. [26]. Due to space limitation, we provide details of the algorithm in Appendix C. The intuition is that, if we have two estimations of the same value, their (weighted) average would be closer to the true value.

Knowing that the noisy estimates can be decomposed into true values and pure noise, our method generates all required noise in advance, followed by the consistency enforcement. In this way, the consistent off-line noise can be directly added to the incoming true values during online publishing. Because of the consistency of both the true values and the noise, the noisy estimates will also be consistent. Moreover, we prove that the result of our online algorithm is equivalent to that of the off-line algorithm (the proof is deferred Appendix C).

Theorem 3.1. *The online consistency algorithm gives identical results as the off-line consistency algorithm.*

This together with the fact that the off-line consistency algorithm can be seen as post-processing and thus satisfies DP, we can argue that our online algorithm also satisfies DP.

3.5 Smoother

In this section, we introduce a smoother to further improve the utility of the algorithm. Assuming the hierarchy from perturber has h layers, the smoother is designed to replace the values from the first s lower-level layers with predictions, which are based on previous estimations. The first question is how to choose s.

Optimizing s. Selecting s is important. A larger s results in smaller noise errors: because there are now $h - s$ layers in the hierarchy, each layer will receive more privacy budget according to sequential composition (given in Section 2.4). On the other hand, a larger s probably leads to a larger bias (because we are only doing the actual estimate once every b^s values; other estimates are from predictions based on previous values, thus are independent of the true values and less accurate). Choosing a good value of s thus is a balance between noise errors and bias. Note that we already have noise error term from Equation 4, but we need to calculate the bias introduced by the smoother (the truncation bias in Equation 4 already exists and does not change with s).

To estimate the smoothing bias, we assume that for each value, the bias amount is approximately $\theta/3$. We then assume there are approximately $b^s/2$ values in a query. Then the average squared bias is approximated by $\frac{b^{2s}}{4}\frac{\theta^2}{9}$. Therefore, we use the following equation to approximate the squared error:

$$(b-1)\left(\log_b(r) - s\right)^3 \frac{2\theta^2}{\epsilon^2} + \frac{b^{2s}}{4}\frac{\theta^2}{9}. \tag{7}$$

Given ϵ and r, s can be computed by minimizing the above error.

Smoothing Method. Given s, we now describe choices of implementing the smoother. We consider a set of methods proposed in the literature, and present their details in Appendix D. Among them, the most straightforward one is "Recent" smoother, which predicts the next values based on the most recent estimation. In evaluation, we find it works the best, probably because the dataset we use is spiky.

3.6 Summary and Discussions

In summary, our method takes the raw stream $V = \langle v_1, v_2, \ldots \rangle$ as input and outputs a private stream $\tilde{V} = \langle \tilde{v}_1, \tilde{v}_2, \ldots \rangle$. Algorithm 1 gives the details of our method: We first cache the first m values and obtain θ. Then for each of the following values, we first truncate it and then use the hierarchical method together with the smoother to output the noisy value.

In this paper, we focus on the setting used in PAK, where the threshold optimizer does not publish the first m values, but uses them to obtain θ. After the first m values, it sends θ to the perturber and truncates any incoming value by θ. The perturber then outputs values using the hierarchical method, and there is a smoother that further processes the result. Our method is also flexible and can work in other settings. We will discuss more about the flexibility of ToPS in Section 7.

We claim that ToPS satisfies ϵ-DP. The perturber uses ϵ/h to add Laplace noise to each layer of the hierarchical structure. By sequential composition, the overall data structure satisfies ϵ-DP. To find the threshold, ToPS uses a disjoint set of m observations and runs an ϵ-DP algorithm. Due to the parallel composition property of DP, the threshold optimizer and the perturber together satisfy ϵ-DP. The online consistency algorithm and the smoother's operations are post-processing procedures and do not affect the privacy guarantee.

4 PUBLISHING STREAMS IN LDP SETTING

In this section, we introduce ToPL for publishing streaming data under local DP (LDP). To the best of our knowledge, this is the first algorithm that deals with this problem under LDP.

In LDP, users perturb their values locally before sending them to the server, and thus do not need to trust the server. Applying to the streaming values in our setting, each value should be perturbed before being sent to the server. What the server does is only post-processing of the perturbed reports.

Algorithm 1: ToPS

Input: $V = \langle v_1, v_2, \ldots \rangle$, ϵ, m, upper bound on query range r
Output: $\tilde{V} = \langle \tilde{v}_1, \tilde{v}_2, \ldots \rangle$

$V_m \leftarrow \langle v_1, \ldots v_m \rangle$; // Cache the first m values
for $\theta = 1$ *to* B **do**
  $m_\theta \leftarrow |\{i \mid v_i \le \theta, i \in [m]\}|$; // Equation 5
  $q_\theta(V_m) \leftarrow -\frac{m\theta}{20r\epsilon}\sqrt{2(b-1)\log_b^3(r)} - m_\theta$; // Equation 6
  $\tilde{q}_\theta(V) = q_\theta(V_m) + \text{Lap}\left(\frac{1}{\epsilon}\right)$; // $\text{GS}_q = 1$, q is monotonic
$\theta \leftarrow \arg\max_\theta \tilde{q}_\theta(V)$; // Find θ via Noisy Max (Section 2.3)
/* The previous part of the code finds θ. */
/* Now we are ready to release the stream. */
$h \leftarrow \log_{16} r$;
$s \leftarrow \arg\min_s \left[15\left(\log_{16}(r) - s\right)^3 \frac{2\theta^2}{\epsilon^2} + \frac{16^{2s}}{4}\frac{\theta^2}{9}\right]$; // Equation 7
$u \leftarrow \frac{16^s\theta}{2}$;
$build \leftarrow$ True ; // Indicator to build a tree
foreach $i > m$ **do**
  $v_i \leftarrow \min(v_i, \theta)$; // Truncate
  if *build* **then**
    Init an $(h-s)$-layer hierarchy with fan-out 16 ;
    Assign 0 to all nodes ; // Build the virtual tree
    Add $\text{Lap}\left(\frac{h-s}{\epsilon}\right)$ to each node ;
    Make the tree consistent ; // Appendix C
    $cur\_node \leftarrow$ left-most noisy node on tree ;
    $build \leftarrow$ False ;
  if $(i-m) \bmod r = 0$ **then**
    $build \leftarrow$ True ; // Time to build another tree
  if $(i-m) \bmod 16^s = 0$ **then**
    Output $cur\_node - u \times (16^s - 1)$;
    $u_t \leftarrow cur\_node$;
    $cur\_node \leftarrow$ next noisy node on tree ;
  else
    Output $u/16^s$; // ''Recent'' smoother in Appendix D
    $cur\_node \leftarrow cur\_node + v_i$;

ToPL follows the design framework of ToPS. There is a threshold optimizer to find the threshold based on the optimal estimated error, and the threshold is used to truncate the users' values in the later stage. Different from the centralized DP setting, in the local setting, the obtained threshold will be shared with the users so that they can truncate their values locally. The perturber section is also run within each user's local side, because of the privacy requirement that no other parties other than the users themselves can see the true data. There is no smoother section. In what follows, we describe the construction for the threshold optimizer and the perturber.

4.1 Design of the Threshold Optimizer

In LDP, each user only has a local view (i.e., they only know their own data; no one has a global view of the true distribution of all data), thus there is no Noisy Max mechanism (NM) (described in Section 2.3) that we can use as in the DP setting. Instead, most existing LDP algorithms rely on frequency estimation, i.e., estimation of how many users possess each value, as what the Laplace mechanism does in DP. We also rely on the frequency estimation to find the optimal threshold. Although the distribution estimation is more informative, it is actually less accurate than the Noisy Max mechanism because (to publish more information) more noise needs to be added.

Frequency Estimation in LDP. Li et al. [33] propose the Square Wave mechanism (SW for short) for ordinal and numerical domains. It extends the idea of Randomized Response [49] in that values near the true value will be reported with high probability, and those far from it have a low probability of being reported. The server, after receiving the reports from users, runs a specially designed Expectation Maximization algorithm to find an estimated density distribution that maximizes the expectation of observing the output. For completeness, we describe details about SW in Appendix E.1.

Optimized Threshold with Estimated Distribution. To find the threshold, the baseline method is to find a specific percentile as the threshold θ. This method is used for finding frequent itemset [45]. Based on the lessons learned from the threshold optimizer in the DP setting, we use the optimization equation given in Equation 3 to find θ.

Specifically, denote $\tilde{f}$ as the estimated distribution where $\tilde{f}_t$ is the estimated frequency of value t. Here the set of all possible t to be considered can no longer be $[B] = \{1, 2, \ldots, B\}$. Instead, we sample 1024 values uniformly from $[B]$. This is because SW uses the Expectation Maximization algorithm, and a large domain size makes it time- and space-consuming. Similar to Equation 4 considered in the DP setting, we use an error formula:

$$\frac{r}{3} \cdot \text{Var}[\tilde{v}] + \frac{r^2}{24}\left(\sum_{\theta < t < B} \tilde{f}_t(t-\theta)\right)^2. \tag{8}$$

Here $\text{Var}[\tilde{v}]$ denotes the variance of estimating v, which we will describe later. It is multiplied by $\frac{r}{3}$ because in expectation, a random range query will involve $\frac{r}{3}$ values, and each of them is estimated independently. For the second part of Equation 8, it can be calculated directly with SW. The multiplicative coefficient $\frac{r^2}{24}$ is the averaged case over all possible range queries. That is, denote j as the range of a query, there are $r - j + 1$ range-j queries within a limit r. In total, there are $\sum_{j=1}^{r}(r-j+1)$ possible queries. For each of them, we have a j^2 coefficient in the squared bias. Thus, we have $\frac{\sum_{j=1}^{r}(r-j+1)j^2}{2r(r+1)} = \frac{(r+1)(2r+1)}{12} - \frac{r(r+1)}{8} \approx \frac{r^2}{24}$ as the average-case coefficient.

Using SW as a White Box. To find a reasonable threshold using SW, we make the following modifications. First, we eliminate the smoothing step from SW, because we observe that in some cases, smoothing will "push" the estimated probability density to the two ends of the range. If some density is moved to the high end, the chosen threshold θ can be unnecessarily large.

Second, we add a post-processing step to prune the small densities outputted by SW. In particular, we find the first qualified value w, whose next 5 consecutive estimates are all below 0.01%. This is a signal that the density after w will converge to 0. We thus replace the estimated density after w with 0. In the experiment, we observe that the two steps help to find a more accurate θ.

4.2 Design of the (Local) Perturber

After obtaining the threshold θ, the server sends θ to all users. When a user reports a value, it will first be truncated. The user then reports the truncated value using the Hybrid mechanism.

Table 1: Dataset Characteristics

Dataset	n	max	p_{85}	p_{95}	$p_{99.5}$	p_{100}	avg
DNS	1141961	2000	63	85	135	617	37.9
Fare	8704495	30000	440	1036	2037	26770	279.9
Kosarak	990002	41270	10	28	133	2498	8.1
POS	515597	1657	13	21	39	165	7.5

The method is described in Appendix E.2. It can estimate v with worst-case variance given in Equation 14, which can be plugged into Equation 8 to find θ. Note that the reports are unbiased by themselves. So to answer a range query, we just need to sum up values from the corresponding range, and there is no need for a smoother.

5 EXPERIMENTAL EVALUATION

The experiment includes four phases. First, we give a high-level end-to-end evaluation of the whole process. Second, we evaluate the performance of the hierarchical method with a fixed truncation threshold. Third, we fix the hierarchical method and test different algorithms that give the threshold. Fourth, we evaluate the performance in the local setting.

5.1 Evaluation Setup

Datasets. A total of four real-world datasets are examined.

- DNS: This dataset is extracted from a set of DNS query logs collected by a campus resolver with all user ids and source IP addresses removed [1]. It includes 14 days of DNS queries. The network administrator can use the number of queries to assess Internet usage in a region. We take number of queries as the stream in the evaluation.
- Fare [2]: New York City taxi travel fare. We use the Yellow Taxi Trip Records for January 2019.
- Kosarak [1]: A dataset of clickstreams on a Hungarian website that contains around 10^6 users and 41270 categories. We take it as streaming data and use the size of click categories as the value of the stream.
- POS [56]: A dataset containing merchant transactions of half a million users and 1657 categories. We use the size of the transaction as the value of the stream.

Table 1 gives the distribution statistics of the datasets.

Metrics. To evaluate the performance of different methods, we use the metric of Mean Squared Error (MSE) to answer randomly generated queries. In particular, we measure

$$\mathrm{MSE}(Q) = \frac{1}{|Q|} \sum_{(i,j)\in Q} \left[\tilde{V}(i,j) - V(i,j)\right]^2. \tag{9}$$

where Q is the set of the randomly generated queries. It reflects the analytical utility measured by Equation 1 from Section 2.1 (to demonstrate the actual accuracy, we also have results for mean absolute error in Appendix F). We set $r = 2^{20}$ as the maximal range of any query.

[1]The data collection process has been approved by the IRB of the campus.

Methodology. The prototype was implemented using Python 3.7.3 and NumPy 1.15.3 libraries. The experiments were conducted on servers running Linux kernel version 5.0 with Intel Xeon E7-8867 v3 CPU @ 2.50GHz and 576GB memory. For each dataset and each method, we randomly choose 200 range queries and calculate their MSE. We repeat each experiment 100 times and report the result of mean and standard deviation. Note that the standard deviation is typically very small, and barely noticeable in the figures.

5.2 End-to-end Comparison

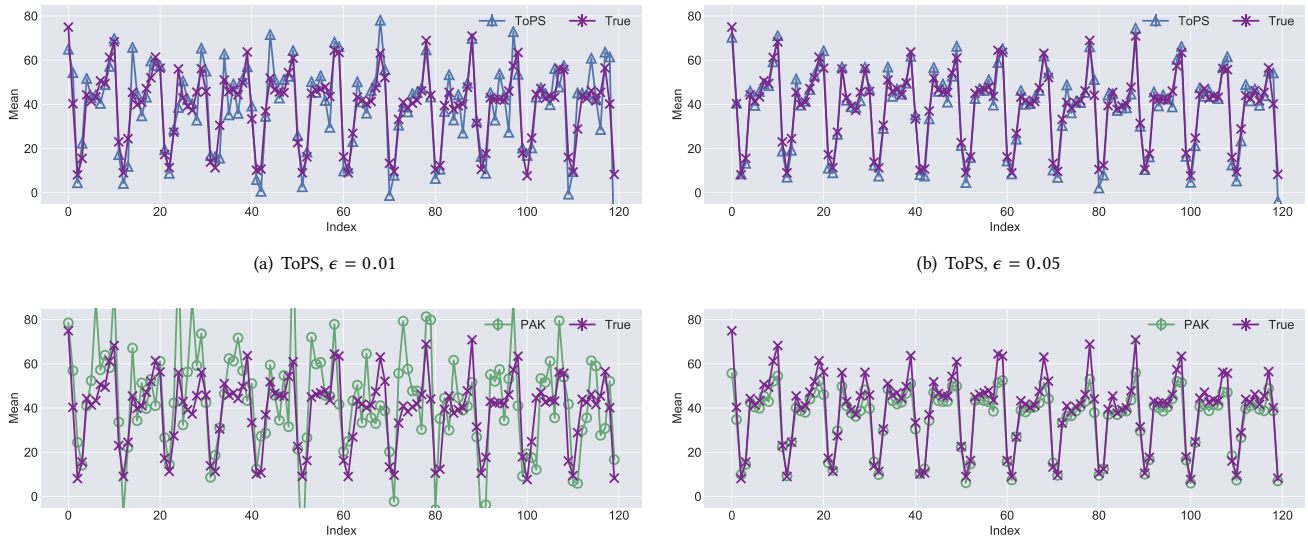
First, as a case study, we visualize the estimated stream of our method ToPS on the DNS dataset (Figure 2). The top row shows the performance of ToPS while the bottom row shows that of PAK. We run algorithms once for each setting to demonstrate the real-world usage. Similar to the setting of PAK, in ToPS, we use the first $m = 65,536$ observations to obtain the threshold θ (we will show later that ToPS does not need this large m observations to be held). Figure 2 indicates that our method ToPS can give fairly accurate predictions when ϵ is very small. On the other hand, PAK, though under a larger ϵ, still performs worse than ToPS. Note that to obtain the threshold θ, PAK satisfies (ϵ, δ)-DP while our ToPS satisfies pure ϵ-DP during the whole process.

We then compare the performance of ToPS and PAK with our metric of MSE given in Equation 9, and show the results in Figure 3. Between ToPS and PAK, we also include two intermediate methods that replace Phase 1 (finding θ) and Phase 2 (hierarchical method) of PAK by our proposed method NM-E (used in threshold optimizer) and $\hat{H}^c_{16}$ (used for the perturber and smoother together), respectively, to demonstrate the performance boost due to our new design (we will evaluate the two phases in more details in later subsections). From the figure, we can see that the performance of all the algorithms gets better as ϵ increases, which is as expected. Second, our proposed ToPS can outperform PAK by 7 to 11 orders of magnitude. Third, the effect (in terms of improving utility) using NM-E is much more significant than using $\hat{H}^c_{16}$. Interestingly, the performance of ToPS and NM-E is similar in the Fare and Kosarak datasets. This is because in these cases, the bias (due to truncation by θ) is dominant.

5.3 Comparison of Stream Publication Phase

Several components contribute to the promising performance of ToPS. To demonstrate the precise effect of each of them, we next analyze them one by one in the reverse order. We first fix other configurations and compare different smoothers and perturbers. Subsequently, we analyze the methods of obtaining the threshold θ in Section 5.4. To make the comparison clear, we set θ to be the 95-th percentile of the values. Moreover, we assume the true values are no larger than θ (the ground truth is truncated). We will compare the performance of different methods in obtaining θ in Section 5.4.

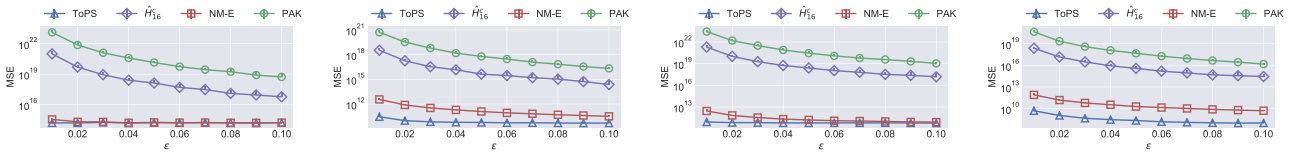
Comparison of Different Smoothers. Fixing a threshold θ and the hierarchical method optimized in Section 3.4, we now compare the performance of five smoother algorithms listed in Section 3.5 (note that the smoothers will replace the 16^s values where s is given in Equation 7). Figure 4 shows the MSE of the smoothers given ϵ from 0.01 to 0.1. As ϵ increases, that is, privacy budget loosens,



(a) ToPS, $\epsilon = 0.01$ (b) ToPS, $\epsilon = 0.05$

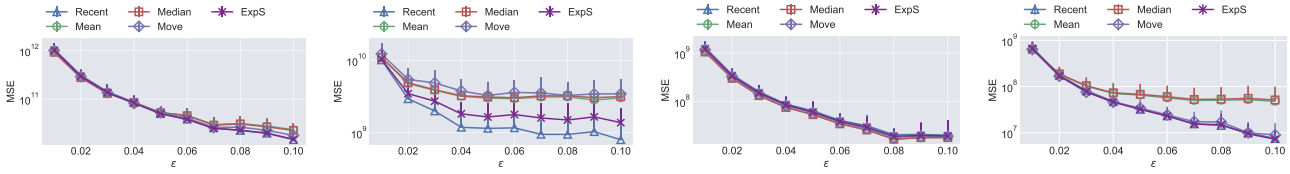
(c) PAK, $\epsilon = 0.1$ (d) PAK, $\epsilon = 0.5$

Figure 2: Visualizations of the DNS stream. The x-axes correspond to time (we partition the 14-day timeframe into 120 intervals, so each point corresponds to the mean of roughly 9000 values or 1.4 hours), and y-axes denotes the moving average. Our ToPS at $\epsilon = 0.01$ can output predictions that are pretty close to the ground truth. PAK gives noisier result even with larger ϵ values.



(a) Fare (b) DNS (c) Kosarak (d) POS

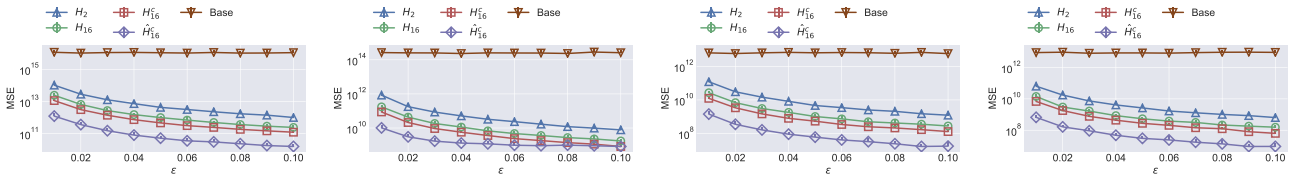
Figure 3: Comparison between PAK and ToPS when answering range queries. We also include two intermediate methods NM-E (our proposed threshold optimizer) and $\hat{H}^c_{16}$ (our proposed the perturber and smoother) that replace the corresponding two phases of PAK to demonstrate the performance boost due to our new design.



(a) Fare (b) DNS (c) Kosarak (d) POS

Figure 4: Evaluation of different smoothing techniques. We vary ϵ from 0.01 to 0.1 in the x-axis. The y-axis shows the query accuracy (MSE).

the overall performance improves although the difference is very small in Fare and Kosarak datasets. This is because there is no clear pattern in these datasets. In the DNS dataset, Recent performs better than others, as the data is stable in the short term. In POS, the method of Mean and Median performs worse than the other three. This is because Mean and Median consider all the history (all the previous u_i values given from the hierarchy, as described in Section 3.5), while the other methods consider more recent results. Methods that utilize the recent output (i.e., the more recent u_i) will perform better due to the stability property in the dataset (similar



(a) Fare (b) DNS (c) Kosarak (d) POS

Figure 5: Evaluation of different methods of outputting the stream. We vary ϵ from 0.01 to 0.1 in the x-axis and plot the query accuracy (MSE) in the y-axis.

to the case of DNS). Since Recent performs the best in DNS, and is among the best in other datasets, we use it as the default smoother algorithm.

Note that large MSE does not always mean poor utility. Here MSE is large because (1) the original values are large (e.g., >1000), (2) the range query takes the sum, and (3) the square operation further enlarges the values. We also have results for mean absolute error (MAE) and mean query results to better demonstrate the utility in Appendix F.

Comparison of Different Hierarchical Algorithms. To demonstrate the precise effect of each design detail, we line up several intermediate protocols for the hierarchy and threshold, respectively. For the hierarchy component, we evaluate:

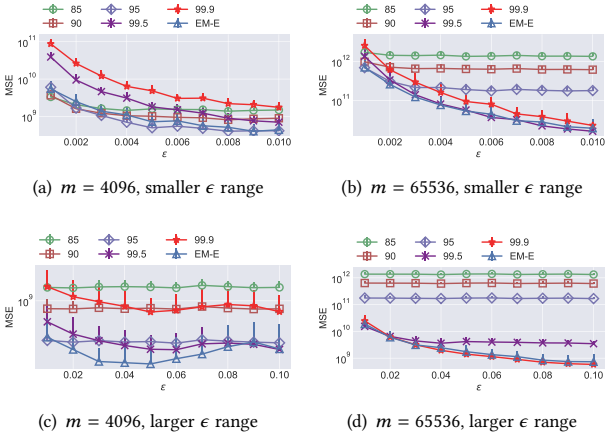
- H_2: Original binary tree used in PAK.
- H_{16}: The optimal fan-out $b = 16$ is used in the hierarchy.
- H^c_{16}: H_{16} with consistency method.
- $\hat{H}^c_{16}$: We use the hat notation to denote the Recent smoother. It is built on top of H^c_{16}.
- Base: A baseline method that always outputs 0. It is used to understand whether a method gives meaningful results.

Figure 5 gives the result varying ϵ from 0.01 to 0.1. First of all, all methods (except the baseline) yield better accuracy as ϵ increases, which is as expected. Moreover, the performance of all methods (except $\hat{H}^c_{16}$) increases by a factor of 100× when ϵ increases from 0.01 to 0.1. This observation is consistent with the analysis that variance is proportional to $1/\epsilon^2$. Comparing each method, using the optimal branching factor (H_{16} versus H_2) can improve the performance by 5×. Moreover, we have approximately another 2× (H^c_{16} versus H_{16}) of accuracy boost by adopting the consistency algorithm. For $\hat{H}^c_{16}$, a constant 10× improvement can be observed over H^c_{16} except in the DNS dataset, where $\hat{H}^c_{16}$ performs roughly the same as H^c_{16} when $\epsilon > 0.08$. The reason is that the error of $\hat{H}^c_{16}$ is composed of two parts. One is the noise error from the constraint of DP, the other the bias error of outputting the predicted values. When ϵ is large, the bias dominates the noise in the DNS dataset.

5.4 Comparison of Threshold Phase

After examining the performance of different methods of outputting the stream, we now switch gear to look at the algorithms for finding the threshold.

Setup. Following PAK's method [36], we use the first m values to obtain the threshold. To eliminate the unexpected influence of distribution change, we use the same m values for now to build the hierarchy using $\hat{H}^c_{16}$ (with the best smoothing strategy called Recent smoother).



(a) $m = 4096$, smaller ϵ range (b) $m = 65536$, smaller ϵ range

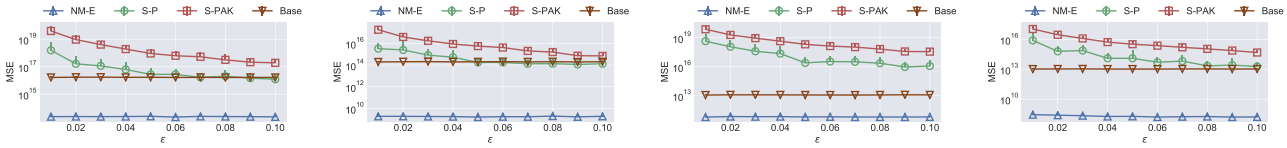
(c) $m = 4096$, larger ϵ range (d) $m = 65536$, larger ϵ range

Figure 6: MSE of answering range queries using $\hat{H}^c_{16}$ on DNS dataset. The true p-th percentile for different p values are evaluated. We also include NM-E, which uses $\epsilon = 0.05$.

No Single Quantile Works Perfectly for All Scenarios. To show that no single p-quantile can work perfectly for all scenarios, we choose the true p-quantile for $p \in \{85, 90, 95, 99.5, 99.9\}$ and test in different scenarios (with different ϵ and m values). We also include our threshold optimizer (NM-E) which is introduced to find a threshold only based on the estimated error and set $\epsilon = 0.05$ for it. Figure 6 shows the results of answering range queries given these true percentiles and it reveals several findings. First, the performance improves as ϵ increases for all p values. Second, in some cases, the performance improvement is negligible with respect to ϵ (e.g., $p = 80$ and 85 in Figure 6(b) and $p = 85$, 90 and 95 in Figure 6(d)). This is because p is too small in these scenarios, which makes the bias dominate the noise error. Third, our threshold optimizer with $\epsilon = 0.05$ can achieve similar performance with the optimal p-quantile.

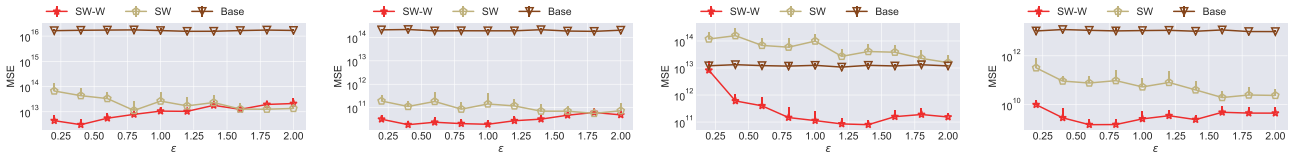
Varying ϵ. We then compare three methods that output θ:

- NM-E: The threshold optimizer in ToPS. It does not require a percentile.



(a) Fare (b) DNS (c) Kosarak (d) POS

Figure 7: Evaluation of different methods to find the threshold θ. We vary ϵ from 0.01 to 0.1 in the x-axis. The y-axis shows the MSE of answering range queries using $\hat{H}^c_{16}$ (to make comparison clear, we use a fixed $\epsilon = 0.05$ for it). Base is a baseline method that always outputs 0.



(a) Fare (b) DNS (c) Kosarak (d) POS

Figure 8: LDP evaluation of different methods of outputting the threshold. We vary ϵ from 0.2 to 2 in the x-axis. The y-axis shows the query accuracy (MSE).

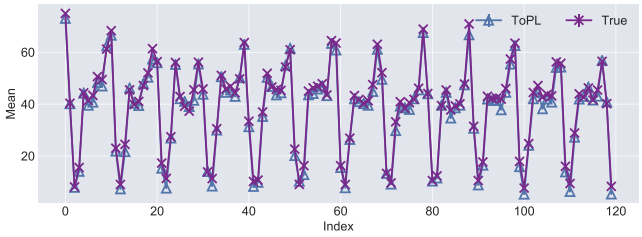


Figure 9: Visualizations of the DNS stream. The x-axes correspond to the time, and the y-axes denote the moving average. Our ToPL at $\epsilon = 1$ can output predictions that are pretty close to the ground truth.

- S-PAK: The smooth sensitivity method used by PAK. We use $p = 99.5$, as used by PAK.
- S-P: The original smooth sensitivity method. Similar to S-PAK, we also use $p = 99.5$.

In Figure 7, we compare with existing differentially private methods on finding the threshold θ. We vary the value of ϵ for obtaining θ, and use $\hat{H}^c_{16}$ to answer range queries. Note that to make the comparison clearer, we fix $\epsilon = 0.05$ in $\hat{H}^c_{16}$. In all the datasets, our proposed NM-E performs much better than existing methods in terms of MSE. Moreover, the performance does not change much when ϵ increases. The reason is that the output of NM-E is stable even with small ϵ. Finally, both S-PAK and S-P perform worse than the baseline method, which always give 0 regardless of input values, indicating the θ given by them is too large.

5.5 Performance of ToPL

In this section, we evaluate the LDP algorithm ToPL. We first check the methods to find θ. Following the setting of ToPS, we use the first $m = 65,536$ observations to obtain the threshold θ. The difference from the DP setting is that we vary ϵ in a larger range (from 0.2 to 2) due to the larger amount of noise of LDP. Our method finds θ by using the Square Wave (SW) mechanism to estimate the distribution, and then minimizing Equation 8. Moreover, our approach (the final part of Section 4.1) modifies SW to exploit the prior knowledge that the distribution is skewed. We use SW-W to denote this method. In addition, we include another method for comparison, which uses SW as a black box (and we use SW to denote it).

Figure 8 shows the performance of finding θ in ToPL. We vary the ϵ used to find θ from 0.2 to 2 while fixing ϵ used to output the stream to 1. The performance of SW improves with ϵ in all four datasets. The performance of our method is less stable and not improving with ϵ, but SW-W can still outperform SW as well as the baseline in all the four datasets. In the Fare and DNS datasets, the advantage of SW-W is not significant when $\epsilon \geq 1.6$. But in the Kosarak and POS datasets, the performance of SW-W can be as large as 3 orders of magnitude (measured by MSE of answering random range queries) compared to SW.

In Figure 9, we visualize the estimated stream of our method ToPL on the DNS dataset using $\epsilon = 1$. We run algorithms only once to demonstrate real-world usage. Clearly, ToPL and ground truth are on similar trajectories in the figure which means our method ToPL can give pretty accurate predictions.

6 RELATED WORK

6.1 Dealing with Streaming Data in DP

A number of solutions have been proposed for solving the problem of releasing real-time aggregated statistics under differential privacy (DP). Here, besides the PAK's approach [36], we briefly describe several other related works.

Event-level DP. The first line of work is the hierarchical method for handling binary streams, proposed concurrently by Dwork et al. [18] and Chan et al. [8]. Event-level DP is satisfied in this case. Both works assumed each value in the stream is either 0 or 1 and proposed a differentially private continual counter release algorithm over a fixed-length binary stream with a bounded error $O\left(\left(\log^{1.5} n\right)/\epsilon\right)$ at each time step, where n is the stream length. In addition, Chan et al. [8] extend the binary tree method to handle unlimited streams. There is also an online consistency method: for each time slot, if its estimation is no bigger than the previous one, output the previous one; otherwise, increment the previous one by 1. We note that this method handles the binary setting, and thus focuses on ensuring each value is an integer, while our method works on the more general non-binary setting and minimizes the overall noise error.

In a follow-up work by Dwork et al. [19], the authors further assumed the number of 1's in the stream is small, and proposed an online partition algorithm to improve over the previous bound. Chen et al. [10] used the similar idea but worked in a different setting (i.e., each value can be any number instead of a binary number). The method partitions the stream into a series of intervals so that values inside each interval are "stable", and publishes the median of each interval. While Chen et al. [10] works in the non-binary setting, similar to our paper and PAK's , it makes two assumptions that lead to a quite different design. First, as mentioned above, Chen et al. [10] made the stability assumption so that partitioning the stream gives better utility; second, it works in a different scenario (i.e., each value is composed of multiple users, and each user contribute at most 1) and the sensitivity is 1, and therefore, the threshold for truncation is not needed.

User-level DP. Compared to Event-level DP which protects against the change of one single event, the user-level definition models the change of the whole data possessed by the user. Because the user-level DP is more challenging, proposals under this setting rely more on the auxiliary information. In particular, Fan et al. [24] proposed to release perturbed statistics at sampled timestamps and uses the Kalman filter to predict the non-sampled values and correct the noisy sampled values. It takes advantage of the seasonal patterns of the underlying data. Another direction is the offline setting, where the server has the global view of all the values first, and then releases a streaming model satisfying DP. In this setting, Acs and Castelluccia [3] propose an algorithm based on Discrete Fourier Transform (DFT). Rastogi and Nath [39] further incorporate sampling, clustering, and smoothing into the process.

***w*-event-level DP.** To balance the privacy loss and utility loss between user-level and event-level privacy models, relaxed privacy notions are proposed. Bolot et al. [6] extended the binary tree mechanism on releasing perturbed answers on sliding window sum queries over infinite binary streams with a fixed window size and using a more relaxed privacy concept called decayed privacy. Kellaris et al. [29] propose w-event DP and two new privacy budget allocation schemes (i.e., split ϵ into individual events) to achieve it. More recently, Wang et al. [42] work explicitly for spatiotemporal traces, and improve the schemes of Kellaris et al. by adaptively allocating privacy budget based on the similarity of the data sources; Fioretto and Hentenryck [25] improve the schemes by using a similar approach of Fan et al. [24]: sampling representative data points to add noise, and smoothing to reconstruct the other points.

In our work, we follow the event-level privacy model and dealing with a more extended setting where data points are from a bounded range instead of the binary domain, and publish the stream in an online manner. Moreover, we do not rely on any pattern to exist in the data; and we propose methods for both DP and LDP.

6.2 Dealing with Streams in Local DP

In the local DP setting, most existing work focus on estimating frequencies of values in the categorical domain [5, 23, 43, 47, 53]. These techniques can be applied to other applications such as heavy hitter identification [4, 40, 46] frequent itemset mining [38, 45], multi-dimension data estimation [11, 12, 44, 51, 52, 55]. In the numerical/ordinal setting, previous work [15, 41] mostly focused on estimating mean. Recently, Li et al. [33] proposed the square wave mechanism for the more general task of estimating the density.

There are two methods [22, 27] that deal with streaming data in the user-level LDP, and their data models are different from ours. In particular, Erlingsson et al. [22] assume the users' values are integers. Each user's value can change at most a few times, and each change can be either $+1$ or -1. The authors proposed a hierarchical method (similar to Phase II of PAK) to estimate the average change over time and thus get the average value by accumulating the changes. Joseph et al. [27] assume at each time, the users' values are a sequence of bits drawn from several Bernoulli distributions, and the goal is to estimate the average of all the bits held by all users (or the average of the Bernoulli parameters). The authors proposed a method to efficiently control each user's contribution when the server's guess (previous estimation) is accurate, and thus save privacy budget. The authors extend the method to a categorical setting and use it to find the most frequent value held by the users.

There is also a parallel work [48] that works on w-event, metric-based LDP. The proposed protocol assumes there is a pattern in each user's streaming data, and lets each user sample the turning point to report.

7 CONCLUSION AND DISCUSSION

We have presented a privacy-preserving algorithm ToPS to continually output a stream of observations. ToPS first finds a threshold to truncate the stream using the exponential mechanism, considering both noise and bias. Then ToPS runs an online hierarchical structure and adds noise to the stream to satisfy differential privacy (DP). Finally, the noisy data are smoothed to improve utility. We also design a mechanism ToPL that satisfies the local version of DP. Our mechanisms can be applied to real-world applications where continuous monitoring and reporting of statistics, e.g., smart meter data and taxi fares, are required. The design of ToPS and ToPL are

flexible and have the potential to be extended to incorporate more properties of the data. We list some as follows.

Shorten the Holdout of the Stream. We follow the setting of PAK [36] and use the first m values to output the threshold θ. If we want to start outputting the stream sooner, we can use our Threshold optimizer with only fewer observations to find a rough threshold. During the process of outputting the stream, we can use sequential composition (in Section 2.4) to fine-tune the threshold.

Update θ. We follow the setting of PAK and assume the distribution stays the same. If the distribution changes, we can have the Threshold optimizer run multiple times (using either sequential composition to update θ while simultaneously outputting the stream, or the parallel composition theorem to block some values to update θ).

Utilizing Patterns of the Data. If there is further information, such that the data changes slowly (e.g., the current value and the next one differ in only a small amount), or the data changes regularly (e.g., if the values show some Diurnal patterns), are given, we can potentially utilize that to improve the performance of our method as well.

ACKNOWLEDGEMENTS

This project was supported by NSF 1931443, 2047476, a Bilsland Dissertation Fellowship, a Packard fellowship, and gift from Cisco and Microsoft. The authors are thankful to the anonymous reviewers for their supportive reviews.

REFERENCES

[1] Frequent itemset mining dataset repository. http://fimi.ua.ac.be/data/.
[2] New york taxi trip record data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.
[3] G. Acs and C. Castelluccia. A case study: Privacy preserving release of spatio-temporal density in paris. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1679–1688, 2014.
[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] J. Bolot, N. Fawaz, S. Muthukrishnan, A. Nikolov, and N. Taft. Private decayed predicate sums on streams. In *Proceedings of the 16th International Conference on Database Theory*, pages 284–295. ACM, 2013.
[7] T. H. Chan, K.-M. Chung, B. M. Maggs, and E. Shi. Foundations of differentially oblivious algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2448–2467. SIAM, 2019.
[8] T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *ACM Transactions on Information and System Security (TISSEC)*, 14(3):1–24, 2011.
[9] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.
[10] Y. Chen, A. Machanavajjhala, M. Hay, and G. Miklau. Pegasus: Data-adaptive differentially private stream processing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1375–1388. ACM, 2017.
[11] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, and T. Wang. Privacy at scale: Local differential privacy in practice. In *SIGMOD*, 2018.
[12] G. Cormode, T. Kulkarni, and D. Srivastava. Answering range queries under local differential privacy. *PVLDB*, 2019.
[13] W.-Y. Day, N. Li, and M. Lyu. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, pages 123–138, 2016.
[14] Z. Ding, D. Kifer, T. Steinke, Y. Wang, Y. Xiao, D. Zhang, et al. The permute-and-flip mechanism is identical to report-noisy-max with exponential noise. *arXiv preprint arXiv:2105.07260*, 2021.
[15] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *FOCS*, 2013.
[16] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Minimax optimal procedures for locally private estimation. *Journal of the American Statistical Association*, 113(521):182–201, 2018.
[17] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.
[18] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 715–724, 2010.
[19] C. Dwork, M. Naor, O. Reingold, and G. N. Rothblum. Pure differential privacy for rectangle queries via private partitions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 735–751. Springer, 2015.
[20] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4), 2014.
[21] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60. IEEE, 2010.
[22] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. *arXiv preprint arXiv:1811.12469*, 2018.
[23] Ú. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.
[24] L. Fan and L. Xiong. An adaptive approach to real-time aggregate monitoring with differential privacy. *IEEE Transactions on knowledge and data engineering*, 26(9):2094–2106, 2013.
[25] F. Fioretto and P. Van Hentenryck. Optstream: releasing time series privately. *Journal of Artificial Intelligence Research*, 65:423–456, 2019.
[26] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1), 2010.
[27] M. Joseph, A. Roth, J. Ullman, and B. Waggoner. Local differential privacy for evolving data. In *Advances in Neural Information Processing Systems*, pages 2375–2384, 2018.
[28] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.
[29] G. Kellaris, S. Papadopoulos, X. Xiao, and D. Papadias. Differentially private event sequences over infinite streams. *Proceedings of the VLDB Endowment*, 7(12):1155–1166, 2014.
[30] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment*, 12(11):1371–1384, 2019.
[31] J. Lee, Y. Wang, and D. Kifer. Maximum likelihood postprocessing for differential privacy under consistency constraints. In *KDD*, 2015.
[32] N. Li, W. Qardaji, D. Su, and J. Cao. Privbasis: Frequent itemset mining with differential privacy. *Proceedings of the VLDB Endowment*, 5(11):1340–1351, 2012.
[33] Z. Li, T. Wang, M. Lopuhaä-Zwakenberg, B. Skoric, and N. Li. Estimating numerical distributions under local differential privacy. In *SIGMOD*, 2020.
[34] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin. Private memoirs of a smart meter. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*, pages 61–66, 2010.
[35] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84. ACM, 2007.
[36] V. Perrier, H. J. Asghar, and D. Kaafar. Private continual release of real-valued data streams. *ndss*, 2019.
[37] W. H. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 6(14), 2013.
[38] 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.
[39] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 735–746, 2010.
[40] 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.
[41] N. Wang, X. Xiao, Y. Yang, J. Zhao, S. C. Hui, H. Shin, J. Shin, and G. Yu. Collecting and analyzing multidimensional data with local differential privacy. In *Proceedings of IEEE ICDE*, 2019.
[42] Q. Wang, Y. Zhang, X. Lu, Z. Wang, Z. Qin, and K. Ren. Real-time and spatio-temporal crowd-sourced social network data publishing with differential privacy. *IEEE Transactions on Dependable and Secure Computing*, 15(4):591–606, 2016.
[43] T. Wang, J. Blocki, N. Li, and S. Jha. Locally differentially private protocols for frequency estimation. In *USENIX Security*, 2017.
[44] 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.
[45] T. Wang, N. Li, and S. Jha. Locally differentially private frequent itemset mining. In *SP*, 2018.
[46] T. Wang, N. Li, and S. Jha. Locally differentially private heavy hitter identification. *IEEE TDSC*, 2019.

[47] T. Wang, Z. Li, N. Li, M. Lopuhaä-Zwakenberg, and B. Skoric. Consistent and accurate frequency oracles under local differential privacy. In *NDSS*, 2020.
[48] Z. Wang, W. Liu, X. Pang, J. Ren, Z. Liu, and Y. Chen. Towards pattern-aware privacy-preserving real-time data collection. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 109–118. IEEE, 2020.
[49] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309), 1965.
[50] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson. Differentially private sql with bounded user contribution. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2020.
[51] M. Xu, B. Ding, T. Wang, and J. Zhou. Collecting and analyzing data jointly from multiple services under local differential privacy. *VLDB*, 2020.
[52] J. Yang, T. Wang, N. Li, X. Cheng, and S. Su. Answering multi-dimensional range queries under local differential privacy. *VLDB*, 2021.
[53] M. Ye and A. Barg. Optimal schemes for discrete distribution estimation under locally differential privacy. *IEEE Transactions on Information Theory*, 2018.
[54] C. Zeng, J. F. Naughton, and J.-Y. Cai. On differentially private frequent itemset mining. *Proceedings of the VLDB Endowment*, 6(1):25–36, 2012.
[55] 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.
[56] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 401–406. ACM, 2001.

A SUPPLEMENTARY MECHANISMS OF DP

We review the method of smooth sensitivity that PAK [36] use for estimating the percentile.

Smooth Sensitivity. Rather than the global sensitivity that considers any pair of neighboring sequences, the local sensitivity fixes one dataset V and only considers all possible neighboring sequence V' around V.

$$\mathsf{LS}_V(f) = \max_{V':V'\simeq V} ||f(V) - f(V')||_1.$$

The advantage of using local sensitivity is that we only need to consider neighbors of V which could result in lower sensitivity of the function f, and consequently lower noise added to the true answer f. Unfortunately, replacing the global sensitivity with local sensitivity directly (e.g., in the Laplace mechanism) violates DP. This is handled by using smooth sensitivity [35] instead.

For $b > 0$, the b-smooth sensitivity of f at $V \in \mathcal{V}$, denoted by $\mathsf{SS}_{V,b}(f)$, is defined as

$$\mathsf{SS}_{V,b}(f) = \max_{V'} \left\{ \mathsf{LS}_{V'}(f) \cdot e^{-b\cdot d(V,V')} \right\},$$

where $d(V, V')$ denotes the hamming distance between V and V'. The method of smooth sensitivity is given below:

$$\mathbf{A}(V) = f(V) + \frac{\mathsf{SS}_{V,b}}{a} \cdot Z,$$

where Z is a random variable from some specified distribution. To obtain $(\epsilon, 0)$-DP, Z is drawn from the Cauchy distribution with density $\propto \frac{1}{1+|z|^\gamma}$ for $\gamma > 1$. But to use an exponentially decaying distribution (which gives good accuracy), such as standard Laplace or Gaussian distribution, one can only obtain (ϵ, δ)-DP. For both, we set $b \le \frac{\epsilon}{-2\log(\delta)}$ as the smoothing parameter. If we use the Laplace distribution with scale 1, $a = \frac{\epsilon}{2}$. When the noise is standard Gaussian, then $a = \frac{\epsilon}{\sqrt{-\ln\delta}}$ [35].

B MORE DETAILS ABOUT PAK

PAK computes the smooth sensitivity of the empirical p-quantile, i.e., $\hat{x}_p$, as

$$\mathsf{SS}_{V,b}(\hat{x}_p) = \max_{k=0,1,\ldots,m+1} \left\{ e^{-bk} \cdot \max_{t=0,1,\ldots,k+1} [V^s(P+t) - V^s(P+t-k-1)] \right\}.$$

Here, V^s is the sorted string of the first m values of V in ascending order, where $V^s(i) = 0$ if $i < 1$ and B if $i > m$. And P is the rank of $\hat{x}_p$. After computing the smooth sensitivity, Nissim et al.[35] sets the threshold θ as

$$\theta = \hat{x}_p + \frac{\mathsf{SS}_{V,b}(\hat{x}_p)}{a} \cdot Z,$$

where Z is a random variable from some specified distribution in order to satisfy DP.

PAK [36] propose to bound $\Pr\left[\theta < x_p\right]$ to be arbitrarily small (by an arbitrary β) and thus uses

$$\theta = \hat{x}_p + \frac{\kappa \mathsf{SS}_{V,b}(\hat{x}_p)}{a} \cdot (Z + G_{\text{ns}}^{-1}(1-\beta)),$$

where κ is a positive real number $\left(1 - \frac{(e^b-1)G_{\text{ns}}^{-1}(1-\beta_{\text{lt}})}{a}\right)^{-1}$ and G_{ns} denotes the CDF of the distribution of Z.

The threshold θ released via the above mechanism is differentially private since $\kappa\mathsf{SS}_{V,b}(\hat{x}_p)$ is a smooth upper bound of $\hat{x}_p$ and κ only depends on public parameters.

In their evaluation, the authors aim to get the 99.5-percentile and set $p = 99.575$, $\beta = 0.3 \cdot 0.02$, and $\delta = 1/n^2$.

C CONSISTENCY ALGORITHM

Off-line Consistency [26]. We use x to denote a node on the hierarchy H, and let $\ell(x)$ to be the height of x (the height of a leaf is 1; and root is of height h). We also denote $\mathrm{prt}(x)$, $\mathrm{chd}(x)$, and $\mathrm{sbl}(x)$ to denote the children, parent, and siblings of x, respectively. We use $H(x)$ to denote the value corresponding to node x in the hierarchy. The first step updates the values in H from bottom to top. The leaf nodes remain the same; and for each height-ℓ node x, where ℓ iterates from 2 up to h, we update it

$$H(x) \leftarrow \frac{b^l - b^{l-1}}{b^l - 1} H(x) + \frac{b^{l-1} - 1}{b^l - 1} \sum_{y \in \mathrm{chd}(x)} H(y). \tag{10}$$

We then update the values again from top to bottom. This time the value on root remains the same, and for each height-ℓ node x, where ℓ iterates from $h-1$ down to 1, we update it

$$H(x) \leftarrow \frac{b-1}{b} H(x) + \frac{1}{b}\left(H(\mathrm{prt}(x)) - \sum_{y \in \mathrm{sbl}(x)} H(y) \right). \tag{11}$$

An Online Algorithm. We decompose the noisy values in H into two parts: the true values and the pure noises, denoted as T and N, respectively. N is the independent noise from the Laplace mechanism (described in Section 2.3). T is defined by induction: for a leaf x, $T(x)$ corresponds to one true value v, and for a node x in a higher level, $T(x) = \sum_{y\in\mathrm{chd}(x)} T(y)$. The true values from T are

consistent naturally. Thus if N is consistent, H is also consistent. To see it, consider any internal node x,

$$\begin{aligned} H(x) &= T(x) + N(x) \\ &= \sum_{y \in \text{chd}(x)} T(y) + \sum_{y \in \text{chd}(x)} N(y) \\ &= \sum_{y \in \text{chd}(x)} (T(y) + N(y)) = \sum_{y \in \text{chd}(x)} H(y). \end{aligned}$$

In the online consistency algorithm, we internally generate the noise hierarchy N and run the steps given in Equation 10 and Equation 11 to make N consistent. After this pre-processing step, we can ignore the higher-layers of the hierarchy, and use only the leaf nodes which add consistent noise to each individual value. This is because the results from the higher-layers are already consistent with those from the leaves, thus it suffices to only output the most fine-grained result.

The online consistency algorithm satisfies DP as long as it gives identical output distribution as the offline algorithm [26]. Now we prove this fact. We first restate the theorem (Theorem 3.1):

Theorem C.1. *The online consistency algorithm gives identical results as the off-line consistency algorithm.*

Proof. We first examine the bottom-up update step. According to Equation 10, the updated $N(x)$ equals to

$$\frac{b^l - b^{l-1}}{b^l - 1} N(x) + \frac{b^{l-1} - 1}{b^l - 1} \sum_{y \in \text{chd}(x)} N(y).$$

Adding $T(x)$ to it, we have the updated $H(x)$ equals to

$$\begin{aligned} &\frac{b^l - b^{l-1}}{b^l - 1} N(x) + \frac{b^{l-1} - 1}{b^l - 1} \sum_{y \in \text{chd}(x)} N(y) + T(x) \\ =&\frac{b^l - b^{l-1}}{b^l - 1} (N(x) + T(x)) + \frac{b^{l-1} - 1}{b^l - 1} \left(\sum_{y \in \text{chd}(x)} N(y) + T(x) \right) \\ =&\frac{b^l - b^{l-1}}{b^l - 1} (N(x) + T(x)) + \frac{b^{l-1} - 1}{b^l - 1} \sum_{y \in \text{chd}(x)} (N(y) + T(y)) \quad (12) \\ =&\frac{b^l - b^{l-1}}{b^l - 1} H(x) + \frac{b^{l-1} - 1}{b^l - 1} \sum_{y \in \text{chd}(x)} H(y). \end{aligned}$$

Equation 12 is because of the consistency of T so that $T(x) = \sum_{y \in \text{chd}(x)} T(y)$. This gives the same result as if we run the off-line consistency algorithm. Similarly, during the top-down update step (in Equation 11), we have the updated $N(x)$ equals to

$$\frac{b-1}{b} N(x) + \frac{1}{b} \left(N(\text{prt}(x)) - \sum_{y \in \text{sbl}(x)} N(y) \right).$$

Adding $T(x)$ to it, we have the updated $H(x)$ equals to

$$\begin{aligned} &\frac{b-1}{b} N(x) + \frac{1}{b} \left(N(\text{prt}(x)) - \sum_{y \in \text{sbl}(x)} N(y) \right) + T(x) \\ =&\frac{b-1}{b} (N(x) + T(x)) + \frac{1}{b} \left(N(\text{prt}(x)) - \sum_{y \in \text{sbl}(x)} N(y) + T(x) \right) \\ =&\frac{b-1}{b} H(x) + \frac{1}{b} \left(H(\text{prt}(x)) - \sum_{y \in \text{sbl}(x)} H(y) \right). \quad (13) \end{aligned}$$

which is the same result as if we run the off-line consistency algorithm. Equation 13 also holds because of the consistency of T so that $T(x) = T(\text{prt}(x)) - \sum_{y \in \text{sbl}(x)} T(y)$. □

D SMOOTHING METHODS

Denote $u_1, u_2, \ldots$ as the noisy estimates given by the leaves of the perturber (or the $(s+1)$-th levels of the original hierarchy). Each u_i is the noisy sum of b^s values. Let $u_0 = \frac{1}{2} b^s \theta$ (initially there are no estimations from the hierarchy; we thus use half of the threshold as mean). The smoother will take the sequence of u and output the final result $\tilde{v}_i$ for each input value. Let $t = \lceil i/b^s \rceil$, we consider several functions:

(1) Recent smoother: $\tilde{v}_i = u_t / b^s$. It takes the mean of the most recent output from the perturber.
(2) Mean smoother: $\tilde{v}_i = \frac{1}{b^s t} \sum_{j=0}^{t} u_j$. It takes the mean of the output from the perturber up until the moment.
(3) Median smoother: $\tilde{v}_i = \text{median}(u_1, \ldots, u_t)$. Similar to the mean smoother, the median smoother takes the median of the output from the perturber up until the moment.
(4) Moving average smoother: $\tilde{v}_i = \frac{1}{b^s w} \sum_{j=t+1-w}^{t} u_j$. Similar to the mean smoother, it takes the mean over the most recent w outputs from the perturber. When $t+1 < w$, we use the average of the first $t+1$ values of u divided by b^s as $\tilde{v}_i$.
(5) Exponential smoother: $\tilde{v}_i = \frac{u_0}{b^s}$ if $t = 0$, and $\tilde{v}_i = \alpha \frac{u_t}{b^s} + (1 - \alpha) \tilde{v}_{i-b^s}$ if $t > 0$, where $0 \le \alpha \le 1$ is the smoothing parameter. The exponential smoother put more weight on the more recent values from the hierarchy.

E MECHANISMS OF LOCAL DIFFERENTIAL PRIVACY

In this subsection, we review the primitives proposed for LDP. We use v to denote the user's private value, and y as the user's report that satisfies LDP. In this section, following the notations in the LDP literature, we use p and q to denote probabilities.

E.1 Square Wave Mechanism for Density Estimation

Li et al. [33] propose an LDP method that can give the full density estimation. The intuition behind this approach is to try to increase the probability that a noisy reported value carries meaningful information about the input. Intuitively, if the reported value is the true value, then the report is a "useful signal", as it conveys the extract correct information about the true input. If the reported value is not the true value, the report is in some sense noise that needs to be removed. Exploiting the ordinal nature of the domain, a report that is different from but close to the true value v also carries useful information about the distribution. Therefore, given input v, we can report values closer to v with a higher probability than values that are farther away from v. The reporting probability looks like a squared wave, so the authors call the method Square Wave method (SW for short).

Without loss of generality, we assume values are in the domain of $[0, 1]$. To handle an arbitrary range $[\ell, r]$, each user first transforms v into $\frac{v-\ell}{r-\ell}$ (mapping $[\ell, r]$ to $[0, 1]$); and the estimated result is transformed back. Define the "closeness" measure $b = \frac{\epsilon e^\epsilon - e^\epsilon + 1}{2e^\epsilon(e^\epsilon - 1 - \epsilon)}$, the Square Wave mechanism SW is defined as:

$$\forall y \in [-b, 1+b],\ \Pr[\mathrm{SW}(v) = y] = \begin{cases} p, & \text{if } |v - y| \le b\ , \\ q, & \text{otherwise}\ . \end{cases}$$

By maximizing the difference between p and q while satisfying the total probability adds up to 1, we can derive $p = \frac{e^\epsilon}{2be^\epsilon + 1}$ and $q = \frac{1}{2be^\epsilon + 1}$.

After receiving perturbed reports from all users, the server runs the Expectation Maximization algorithm to find an estimated density distribution that maximizes the expectation of observing the output. Additionally, the server applies a special smoothing requirement to the Expectation Maximization algorithm to avoid overfitting.

E.2 Candidate Methods for the Perturber

As we are essentially interested in estimating the sum over time, the following methods that estimate the mean within a population are useful. We first describe two basic methods. Then we describe a method that adaptively uses these two to get better accuracy in all cases. Our perturber will use the final method.

Stochastic Rounding. This method uses stochastic rounding to estimates the mean of a continuous/ordinal domain [16]. We call it Stochastic Rounding (SR for short). Assume the private input value v is in the range of $[-1, 1]$ (otherwise, we can first projected the domain into $[-1, 1]$), the main idea is to round v to v' so that $v' = 1$ with probability $p_1 = \frac{1}{2} + \frac{v}{2}$ and $v' = -1$ w/p $1 - p_1$. This stochastic rounding step is unbiased in that $\mathbb{E}[v'] = v$. Then given a value $v' \in \{-1, 1\}$, the method runs binary random response to perturb v' into y. In particular, let $p = \frac{e^\epsilon}{e^\epsilon + 1}$ and $q = 1 - p = \frac{1}{e^\epsilon + 1}$, $y = v'$ w/p p, and $y \neq v'$ w/p q. The method has variance $\left(\frac{e^\epsilon + 1}{e^\epsilon - 1}\right)^2 - v^2$.

Piecewise Mechanism. Wang et al. [41] proposed piecewise mechanism. It is also used for mean estimation, but can get more accurate mean estimation than SR when $\epsilon > 1.29$. In this method, the input domain is $[-1, 1]$, and the output domain is $[-s, s]$, where $s = \frac{e^{\epsilon/2} + 1}{e^{\epsilon/2} - 1}$. For each $v \in [-1, 1]$, there is an associated range $[\ell(v), r(v)]$ where $\ell(v) = \frac{e^{\epsilon/2} \cdot v - 1}{e^{\epsilon/2} - 1}$ and $r(v) = \frac{e^{\epsilon/2} \cdot v + 1}{e^{\epsilon/2} - 1}$, such that with input v, a value in the range $[\ell(v), r(v)]$ will be reported with higher probability than a value outside the range. The high-probability range looks like a "piece" above the true value, so the authors call the method Piecewise Mechanism (PM for short). The perturbation function is defined as

$$\forall_{y \in [-s,s]}\ \Pr[\mathrm{PM}(v) = y] = \begin{cases} p = \frac{e^{\epsilon/2}}{2} z, & \text{if } y \in [\ell(v), r(v)] \\ q = \frac{1}{2e^{\epsilon/2}} z, & \text{otherwise.} \end{cases}$$

where $z = \frac{e^{\epsilon/2} - 1}{e^{\epsilon/2} + 1}$. Compared to SR, this method has a variance of $\frac{v^2}{e^{\epsilon/2} - 1} + \frac{e^{\epsilon/2} + 3}{3(e^{\epsilon/2} - 1)^2}$ [41].

Hybrid Mechanism. Both SR and PM incurs a variance that depends on the true value, but in the opposite direction. In particular, when $v = \pm 1$, the variance of SR is lowest, but the variance of PM is highest. Wang et al. [41] thus propose a method called Hybrid Mechanism (HM for short) to achieve good accuracy for any v. In particular, define $\alpha = 1 - e^{-\epsilon/2}$, when $\epsilon > 0.61$, users use PM w/p α and SR w/p $1 - \alpha$. When $\epsilon \le 0.61$, only SR will be called. It is proved by Wang et al. [41] HM gives better accuracy than SR and PM. In particular, the worst-case variance is

$$\mathrm{Var}[\tilde{v}] = \begin{cases} \left(\frac{e^\epsilon + 1}{e^\epsilon - 1}\right)^2, & \text{when } \epsilon \le 0.61 \\ \frac{1}{e^{\epsilon/2}} \left[\left(\frac{e^\epsilon + 1}{e^\epsilon - 1}\right)^2 + \frac{e^{\epsilon/2} + 3}{3(e^{\epsilon/2} - 1)}\right], & \text{when } \epsilon > 0.61. \end{cases} \tag{14}$$

F MORE RESULTS

Denote Q as the set of the 200 randomly generated queries, we show the results of Mean Absolute Error (MAE):

$$\mathrm{MAE}(Q) = \frac{1}{|Q|} \sum_{(i,j) \in Q} \left|\tilde{V}(i, j) - V(i, j)\right|.$$

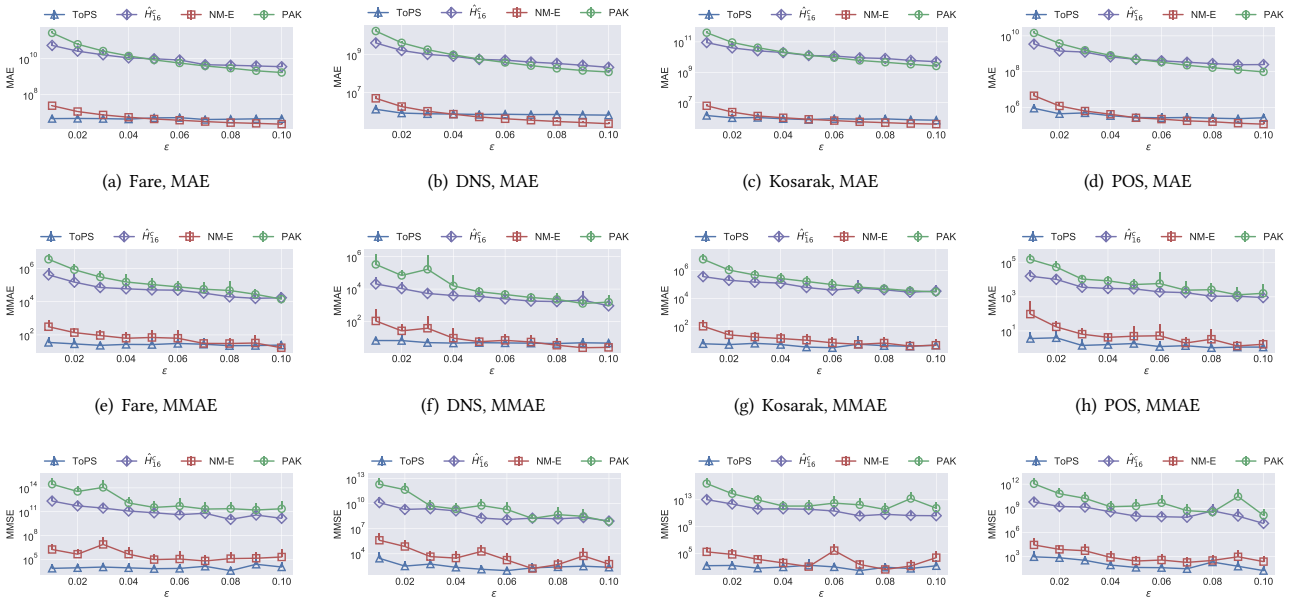
Moreover, we also show results for the normalized results, or the mean of range queries, namely,

$$\mathrm{MMSE}(Q) = \frac{1}{|Q|} \sum_{(i,j) \in Q} \left[\frac{\tilde{V}(i, j)}{j - i} - \frac{V(i, j)}{j - i}\right]^2,$$

$$\mathrm{MMAE}(Q) = \frac{1}{|Q|} \sum_{(i,j) \in Q} \left|\frac{\tilde{V}(i, j)}{j - i} - \frac{V(i, j)}{j - i}\right|.$$

Figure 10 gives results on these metrics. Let us first look at the first row, which gives MAE results. The overall trend is similar to that of MSE (Figure 3). One notable difference is that the better hierarchical method does not give a better overall result (ToPS versus NM-E and $\hat{H}^c_{16}$ versus PAK). This is because the better hierarchical method (especially the consistency step) is optimizing the squared error. We note that Lee et al. [31] proposed methods for optimizing absolute error (L_1 error), and they can be used in our setting if the target is to minimize absolute errors.

For the second and third row of Figure 10, which corresponds to MMAE and MMSE, respectively, we can see the overall trend and the relative performance of different methods are similar to the case of MAE and MSE. The results are less stable though: this is because the range of the query here introduces another factor of randomness. That is, due to the usage of the hierarchy, the larger the range, the smaller the error will be. For the MMAE metric, the results of our proposed ToPS can be as small as 1, while the existing work of PAK [36] gives errors of 10^3 to 10^4, depending on the dataset. Comparing with the flat method, where a Laplace noise on the order of B/ϵ (where B is the upper bound of the data) or θ/ϵ (if some technique of finding θ is applied) is added to each count, ToPS significantly improves over it. Finally, the results of MMSE show similar trends as that of MSE. These evaluation results further confirm the superiority of ToPS regardless of the evaluation metrics.



(a) Fare, MAE (b) DNS, MAE (c) Kosarak, MAE (d) POS, MAE

(e) Fare, MMAE (f) DNS, MMAE (g) Kosarak, MMAE (h) POS, MMAE

(i) Fare, MMSE (j) DNS, MMSE (k) Kosarak, MMSE (l) POS, MMSE

Figure 10: Comparison of different methods on MAE (first row), MMAE (second row) and MMSE (third row).