Journal of Privacy and Confidentiality
Vol. 12 (1) 2022

TPDP 2020

Submitted Mar 2021
Published Jul 2022

Overlook: DIFFERENTIALLY PRIVATE EXPLORATORY VISUALIZATION

PRATIKSHA THAKER, MIHAI BUDIU, PARIKSHIT GOPALAN, UDI WIEDER, AND MATEI ZAHARIA

Stanford University
e-mail address: prthaker@stanford.edu

VMware Research
e-mail address: mbudiu@vmware.com

VMware Research
e-mail address: pgopalan@vmware.com

VMware Research
e-mail address: uwieder@vmware.com

Stanford University
e-mail address: matei@cs.stanford.edu

Abstract. Data exploration systems that provide differential privacy must manage a privacy budget that measures the amount of privacy lost across multiple queries. One effective strategy to manage the privacy budget is to compute a one-time private synopsis of the data, to which users can make an unlimited number of queries. However, existing systems using synopses are built for offline use cases, where a set of queries is known ahead of time and the system carefully optimizes a synopsis for it. The synopses that these systems build are costly to compute and may also be costly to store.

We introduce Overlook, a system that enables private data exploration at interactive latencies for both data analysts and data curators. The key idea in Overlook is a *virtual synopsis* that can be evaluated *incrementally*, without extra space storage or expensive precomputation. Overlook simply executes queries using an existing engine, such as a SQL database, and adds noise to their results. Because Overlook's synopses do not require costly precomputation or storage, data curators can also use Overlook to explore the impact of privacy parameters interactively. Overlook offers a visual query interface based on the open source Hillview system. Overlook achieves accuracy comparable to existing synopsis-based systems, while offering better performance and removing the need for extra storage.

Key words and phrases: synopsis, curator, virtual synopsis, interactive visualization, privacy.

1. Introduction

1.1. **Motivation.** Privacy has become a key issue for all organizations that collect personal data, from companies to government entities [47, 4, 30]. After organizations collect a dataset, they would like to make it available to internal data analysis teams, or even expose it to external researchers [47], without leaking a significant amount of information about any individual in the dataset. To be broadly useful, a private data analysis system should support ad-hoc, exploratory queries through familiar interfaces while making it easy for the data curator (the administrator configuring the system) to control the amount of information leaked.

The most widely used framework for reasoning about privacy is differential privacy (DP) [14, 16]. A differential privacy mechanism is a *randomized* algorithm $\mathcal{A}$ such that, for any two databases D_1 and D_2 that differ by exactly a single element (say row), and every possible set S of outputs of the algorithm, it holds that

$$\Pr[\mathcal{A}(D_1) \in S] \leq \exp(\epsilon) \cdot \Pr[\mathcal{A}(D_2) \in S].$$

The probability is taken over the internal randomness of the algorithm, and limits the ability of the data analysis to infer which of the two databases were queried, and thus provides a privacy guarantee at the resolution of a single element. Differential privacy quantifies the privacy cost of a statistical analysis through a *privacy budget* denoted by ϵ; a smaller privacy budget implies more error in the query results but more privacy for the individuals in the dataset.

Although many research systems provide differential privacy [37, 28, 17, 31, 32, 45, 21], these current systems are challenging for organizations to configure and use, especially for ad-hoc exploratory analysis. At a high level, current DP systems fall into two categories:

1. **Systems with per-query budgeting:** Systems such as PINQ [37] and FLEX [28] ask users to select a privacy budget, ε, for each query they execute. The total privacy leakage of the system is then bounded by the sum of these ε values. These systems are complex for both users and data curators to use. Users typically have a limited total privacy budget available, $\varepsilon_{\text{total}}$, and need to decide how to divide it between the queries they submit; when they run out of budget, they can no longer make queries. In addition, two users that collaborate can reveal information proportional to the sum of their budgets, so data curators must carefully limit which users can access the system. Such systems would be unsuitable for exposing a research dataset to the public, for instance [47].
2. **Synopsis-based systems:** Systems such as PrivateSQL [31] generate a *synopsis* data structure that can answer a specific class of queries after taking in a dataset, a description of the query class, and a total privacy budget ε. Users can then query the synopsis arbitrarily many times without revealing additional information beyond the ε budget. These systems are more suitable for exploratory analysis and for public access, but unfortunately, they are also challenging to use. Constructing the synopsis requires solving an expensive optimization problem to minimize the error it will produce for a specific query workload, which can take hours even for a modest dataset, and the synopsis can consume a large amount of space, on par with the original data, making it costly for large datasets.

1.2. **Overview.** In this paper, we present Overlook, a system that makes synopsis-based differential privacy practical for one of the most common types of data analysis: visual exploratory analysis of immutable datasets. Visual query interfaces, such as Tableau [46], are one of the most common ways for organizations to expose data internally, and produce a class of queries that are a good fit for synopsis data structures (mostly counting queries). In Overlook, we seek to make private visual queries accessible to both data users and data curators, by designing a system that lets curators tune a synopsis *interactively* to set privacy parameters, and lets users query data interactively at a similar cost to their existing data analysis infrastructure. Overlook runs as an interposition layer in front of existing analytical engines, such as a standard relational database system that supports the SQL programming language for querying data, enabling organizations to benefit from the scalability and optimizations of these existing engines and to offer private visual query interfaces over existing datasets.

The key idea in Overlook is a *virtual synopsis* data structure that represents the noise that would be added by a classical synopsis algorithm in a highly compressed format using a pseudo-random function (PRF). For any counting query (e.g., counting the users in a dataset by country), Overlook can use the virtual synopsis to compute just the noise that should be added to each tuple in the query result. Overlook simply adds this noise to the results computed by any existing query engine. Thanks to this design, users can run queries at a similar speed to their existing query engine. Likewise, while preparing the dataset, data curators can use Overlook to explore parameters of the virtual synopsis *interactively*, e.g., change the total privacy budget ε or its allocation to various columns, and see its results on several queries. Overlook's synopses are based on the hierarchical histogram mechanism [24, 9], a synopsis design that supports multidimensional queries, and can be tuned by curators to provide different noise levels for different dimensions in the data.

Overlook also offers a rich privacy-aware visual query interface built on virtual synopses, based on the open source Hillview system [1]. In particular, we extend built-in visualizations in Hillview, such as histograms and heatmaps, to display information about the noise introduced by DP. The various visualization choices are all based on the virtual synopsis, so they do not cause any additional privacy leakage. Data visualization has some unique features that make it a good candidate for a synopsis-based approach to DP: most queries can be expressed as (combinations of) count queries, for which there are good synopses. Secondly, the visualization itself introduces errors via the quantization to the pixels in the screen. This inherent approximation may even mask the error introduced by DP, particularly for large datasets. Finally, typically data visualization interfaces already incorporate methods for presenting errors and approximations to the user (e.g., via confidence intervals and error bars). These tools help the user understand the results the differential privacy mechanism produces.

Overlook operates in the setup where the data curator is a trusted party with unrestricted access to the data, who wishes to make the dataset accessible for analysis to an untrusted community of data analysts, while ensuring differential privacy. The data curator must make decisions about the privacy parameters used to build the synopsis. These decisions must balance the privacy of the dataset with utility for the analyst. We refer the reader to figure 1 for a schematic diagram of the setup. We emphasize that in our model data access by the curator is *not* private, and does not consume from the privacy budget.

Overlook provides a visual interface that is meant to address the needs of both the data *curator* as well as the data *analyst*. In *curator mode*, the curator can set various privacy parameters such as bins for categorical features and privacy budgets along different data dimensions, on multiple visual queries. The curator can tune these parameters in creating the virtual synopsis, but once the dataset has been published, the curator may not modify them further, as doing so would violate differential privacy. The Overlook user interface (UI) lets the curator quickly see the impact of adjusting various privacy parameters. This allows the curator to analyze the impact of various budgeting choices on the utility to the analyst. To our knowledge, Overlook is the first system to provide interactive feedback for tuning a DP synopsis.

1.3. **Implementation and Evaluation.** We implement Overlook using the Hillview UI, and develop backends to let it run either over a SQL database or over Hillview's built-in distributed execution engine [1], a high performance in-memory query engine that supports approximate query processing for common visualization queries. In both cases, Overlook benefits directly from the optimizations in the underlying engine. Overlook is an open source project, the code is available at `http://github.com/vmware/hillview`.

We evaluate Overlook against the algorithms for computing synopses in DAWA [32] and PrivateSQL [31], and other algorithms in DPBench [23]. Although many of these algorithms build workload-aware synopses, which are optimized for a specific set of queries [35], we find that Overlook's workload-agnostic virtual synopses offer similar levels of error in query results when given the set of visualization queries as input. The key intuition is that in an exploratory analytics setting with many possible queries, a synopsis that "balances" the noise over the possible queries will perform well, and it is not useful to solve a complex optimization problem [44, 42, 24]. Moreover, Overlook's virtual synopses require no precomputation and minimal storage over the underlying histogram. We show that Overlook achieves the same scaling properties as the underlying database while requiring no more than $2.5\times$ the time required to compute equivalent non-private queries. Overlook requires only a 32-byte key for its synopsis, compared to existing synopses that require kilobytes of space to store and potentially gigabytes of memory to compute.

1.4. **Summary of contributions.**

(1) We present Overlook, the first practical differentially private visual analytics **system** that provides interactive configuration for data curators and simultaneously supports interactive multidimensional histogram queries for users. Overlook uses a standard unmodified database for data storage and query execution, by acting as an interposition layer between the user and the database.
(2) We introduce virtual synopsis, a data structure to represent DP synopses for multidimensional counting queries that can be used incrementally to add noise to a specific query, instead of requiring costly precomputation and storage.
(3) We develop a privacy-aware interactive visualization UI for use by both data users and data curators. Used in conjunction with a novel curation mode where the curator can configure virtual synopsis parameters, the UI lets the curator visualize the effect of various parameter choices interactively.

1.4.1. *Organization of the paper.* The rest of this paper is organized as follows:

- Section 2 describes the general structure of Overlook. It talks about the threat model, the roles of the user and data curator, and the functionality that we wish to provide to each.
- Section 3 defines the data model, privacy definitions and data structures such as the hierarchical histogram synopsis structure that we use.
- Section 4 talks about the need for generating synopses quickly and efficiently, how virtual synopses address this need and how they are implemented in Overlook using a crytographic pseudorandom number generator.
- Section 5 describes the implementation of Overlook, including the UI, the privacy interposition layer and the backend.
- Section 6 talks about the user experience of using Overlook, and contrasts it with the experience of using Hillview, the open source visualization engine that it is built on, focusing on how we try to convey the uncertainty inherent in differentially private visualizations.
- Section 7 details experimental results which evaluate Overlook and substantiate our claims regarding its efficiency, scalability and utility.
- Section 8 surveys related work in the literature.

2. System overview

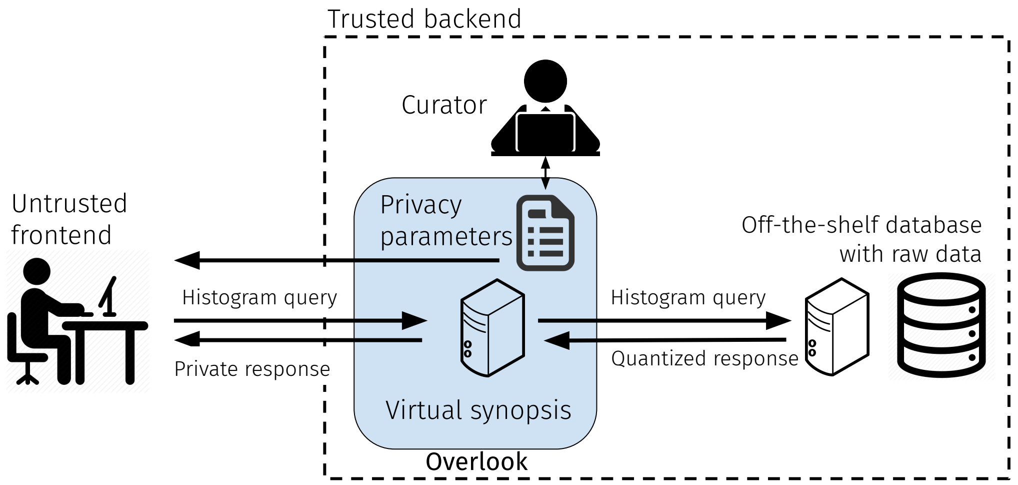


Figure 1. Overlook architecture. Overlook is an interposition layer that sits between a user and an off-the-shelf database. The privacy parameters are created by a trusted data curator; they dictate the privacy policy. The privacy parameters can be read by untrusted users. Overlook rewrites queries into privacy-preserving queries that can be executed by the database, and it sanitizes the answers obtained from the database. Overlook provides the same answers that would be obtained from querying a synopsis of the data, but without computing the actual synopsis.

Figure 1 shows the architecture of Overlook. A data analyst interacts with Overlook through a browser interface that allows them to issue queries to the Overlook root node. The root dispatches the query to the backend, applies a privacy mechanism to the returned result, and returns the private result to the user. The backend servers can run standard off-the-shelf database software, with no special knowledge of DP. This is significant, because DP code is complex and difficult to audit.

The raw data resides on a possibly distributed set of trusted servers; a centralized root receives histogram queries and dispatches them to the servers. The root also stores relevant privacy parameters used to compute the private response to a histogram query. The trusted data curator can make changes to these privacy parameters until the dataset is published, at which point the privacy parameters as well as the dataset must become immutable. The untrusted data analyst can only access published data through the private results of histogram queries. The privacy *parameters* are assumed to be public and visible to both the data curator and the data analyst. The privacy parameters are discussed further in Section 2.3.

Overlook primarily supports *histogram queries.* A histogram query over a column takes as input a set of disjoint buckets and returns the number of data items that fall in each bucket. In Overlook, these counts are perturbed with noise consistent with a differentially-private mechanism, described further in Section 4. Overlook allows users to issue an *arbitrary number* of histogram queries on privately-published data, with no privacy budget restriction. Histogram queries are sufficient to support a large number of visualizations, including histograms, heat maps, pie charts, trellis plots, and cumulative distribution functions (CDFs). The visualizations supported in Overlook are described in Section 6.

2.1. **Threat model.** We assume untrusted users, who can make an unbounded number of queries to the Overlook backend through the Overlook UI. Users may communicate with each other and make queries in parallel or from multiple sessions. Users cannot modify privacy parameters, view raw data, or alter any secret key stored on the root. Side-channel and denial-of-service attacks are out of scope in our work.

The data *curator* is trusted and has access to the raw data and privacy parameters. In *curator mode*, the curator can set various privacy parameters, and visualize the effect of these settings on various queries using the UI. The curator may not modify data or parameters once a dataset has been published, as doing so would violate differential privacy.

The distributed backend is entirely trusted not to leak the data (but otherwise is not DP-aware in any way), including the root server as well as servers hosting the raw data.

2.2. **User interface.** The data curator and data analyst both access Overlook through an interface that is an extension of the Hillview data visualization system [1], which provides a browser interface for interacting with charts and data. There is nothing particularly special about the Hillview UI, we have chosen it because we were familiar with its internals so we could modify it to handle data with uncertainties. In principle any other UI can be adapted to display noisy data. In particular, the UI is completely agnostic to the DP algorithms that run in the root node.

When using the UI in curator mode, the curator may modify privacy parameters and generate new histogram queries under the new parameters prior to publishing the dataset.

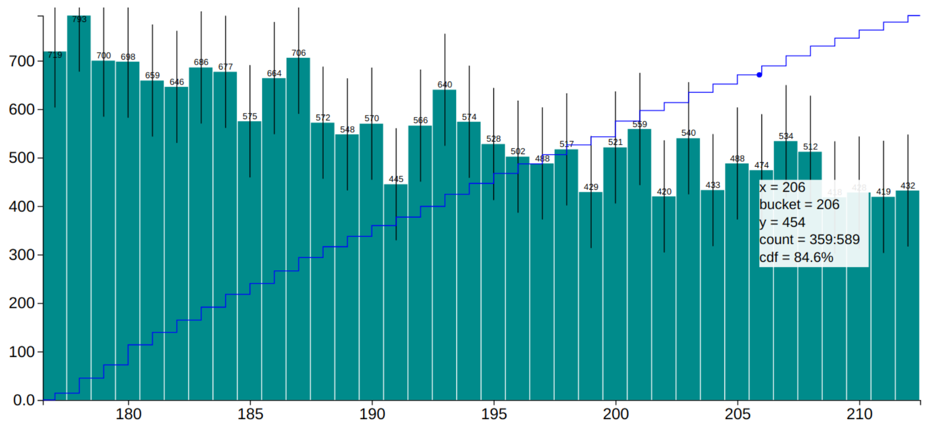
The curator can use the UI in the same way as an analyst might, in order to gauge the effect of various parameter choices on the utility of the analyst.

The result of a histogram query is displayed as an interactive plot. Additional queries can be made by zooming in using the mouse, by selecting an interval, which issues a new histogram query to the backend.

Note that, while the *frontend* is an extension to Hillview, Overlook can be used with any backend that supports count queries. In Section 5, we describe one such alternate implementation using MySQL.

2.2.1. *Supported visualizations.* Overlook essentially supports multi-dimensional histogram visualizations. Multi-dimensional histograms encompass a number of useful visualizations, including degenerate histograms (counts), traditional 1-dimensional histogram, and also heat maps, pie charts, trellis plots, and CDFs.

In addition, the user interface displays schema metadata including standard values such as the column type, but also the privacy policy associated with a column or group of columns.



x = 206
bucket = 206
y = 454
count = 359:589
cdf = 84.6%

FIGURE 2. Histogram plot with CDF curve overlaid. The data at the mouse position is shown in a semi-transparent white rectangle; notice that the bar size (count) is given as an interval, and confidence intervals are plotted for each bar.

Histograms. Overlook's main primitive is a histogram query. This primitive can be applied to create a variety of useful visualizations:

- Histogram queries over a column (with numeric or categorical data). The visual presentation can be a bar chart with confidence intervals, as shown in Figure 2, or, for example, a pie chart, which emphasizes percentage of the whole that falls within each bucket.
- Cumulative distributions functions (CDF) over a column (numeric or categorical). CDF plots are always shown together with a histogram plot. Figure 2 shows a histogram plot with an overlaid CDF curve in dark blue. The CDF is a series of step functions, due to the quantization of the data, as described in Section 2.3.3.

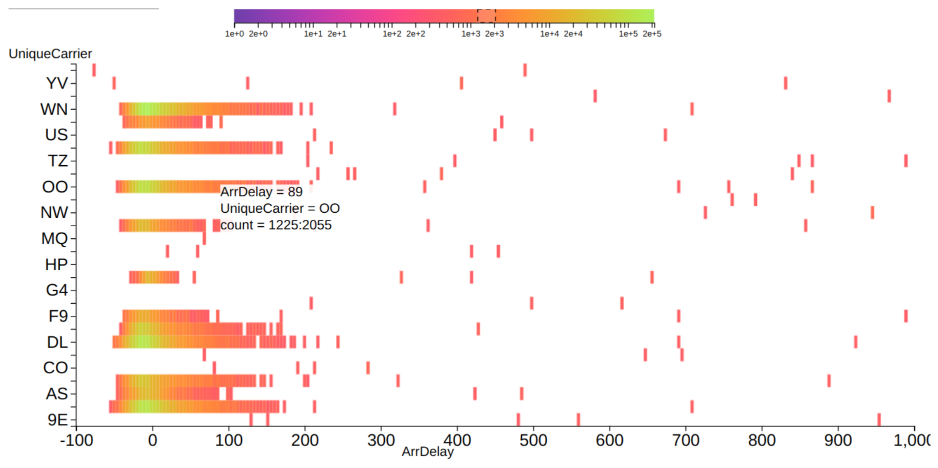


FIGURE 3. Heatmap on two columns. The color shows the count for each combination of values. Values with low confidence are hidden, and the user can highlight with the mouse values within a specific range.

- Histogram queries over a pair of columns, each of which can be either numeric or categorical. This can be visually presented as a heat map as in Figure 3, or for example a trellis plot of 1-dimensional histograms as in Figure 8.

One important feature of Overlook is that it displays estimates of *uncertainty* about the data. For 1-dimensional histograms, this is in the form of 99% confidence intervals.[1] The presentation of uncertainty is discussed further in Section 6.

Counts. In addition to histograms, Overlook supports releasing useful counts ("degenerate histograms") privately, e.g., the number of rows of a table.

2.3. **Curator mode.** The data curator's job is to decide which columns and pairs of columns will be released privately, and to then decide the privacy level for each of those data releases. These parameters are set by the curator in curator mode. Overlook's UI helps the data curator make these decisions. The parameters are currently stored in a JSON file that can only be edited by the curator (but the settings themselves are public). Once the dataset is published, these settings should not be changed, as doing so will violate the differential privacy guarantees.

For each set of columns that is to be released privately, the curator must specify a corresponding *privacy policy*. This policy provides Overlook with information about public values that can be used in the histogram as well as parameters that are used to instantiate the privacy mechanism. The curator can use the Overlook UI to experiment with policy settings and generate sample charts on a dataset before it is published. To our knowledge, Overlook is the first system that provides such functionality to the curator *at interactive speeds*. There are two key attributes of Overlook that makes this possible:

- The ability of Hillview to render visualizations at interactive speeds, even for very large datasets.

[1]Clearly, 99% is just a parameter of our implementation; we have not explored the usability/privacy tradeoffs of this choice.

- The virtual synopses employed by Overlook, which do not need to be materialized separately for each setting of parameters (which might be time and resource intensive), but rather are computed on-the-fly at runtime.

Together these enable the curator to make changes to the privacy parameters, and see the effects of these changes on the visualizations, and thus to concretely visualize the utility tradeoffs incurred by different privacy parameter settings.

Next we describe the parameter settings the curator can choose in a privacy policy. While specifying a policy for *every* such histogram may be impractical, Overlook provides some useful default values for the convenience of the curator.

2.3.1. *Privacy levels.* For each set S of columns that can be visualized together, the curator specifies a corresponding value ε_S that denotes the privacy level that should be used when exploring these columns. A smaller value of ε_S typically results in more privacy at the cost of more noise in the private output. The curator can explore many values of ε_S for each set of columns before deciding which privacy level gives the best tradeoff between data privacy and utility for potential analysts. The curator can specify default values: e.g., "for any three columns use an $\varepsilon_{(*,*,*)} = 0.3$" or specific values for specific column sets, e.g., "for the pair (name,age) use an $\varepsilon_{(\text{name,age})} = 0.1$". The total privacy budget used is $\varepsilon = \sum_S \varepsilon_S$, where each S is a set of columns that can be visualized together.

2.3.2. *Data ranges.* On the first histogram query a user makes for a column, Overlook must return a histogram over a sensible range, after which the user can zoom in to regions of interest. A non-private visualization could compute the true minimum and maximum values in the dataset and return a histogram over the full range. Computing these values in a differentially-private manner is a considerably more complex task [15]. Instead, Overlook requries the data curator to specify publicly-visible values for the initial minimum and maximum for each column in the privacy policy. As in prior work [37, 27], the curator must be careful to not choose values closely associated with specific data points.

2.3.3. *Quantization.* In Overlook, the data curator must specify a public *quantization*, or partitioning, of the data. The need for this is twofold:

(1) The synopsis used in Overlook, detailed in Section 3, operates over a finite, enumerable data domain.
(2) When displaying histograms on categorical data, Overlook uses the curator-specified bin boundaries as public *labels* for the bins.

To illustrate the second point, consider a column that contains the names of patients in a hospital. A non-private histogram might reveal specific names through the choice of bin labels. In Overlook, a curator may set the quantization boundaries to be the letters 'A' through 'Z', so that the finest unit of aggregation is the first letter of the name, and no individual's name is leaked in the published histogram.[2]

[2]While curators should be careful to choose parameters that are public and independent of the data, we note that the curator's decisions may leak information if they are made adaptively based on the true underlying dataset. Preventing this leakage is a challenging issue that is not unique to Overlook, and devising methods to add differential privacy to this kind of human-in-the-loop parameter selection is an interesting avenue for further work.

The idea of public partitioning has been explored in prior systems [37, 27], but the partitioning in those systems has been left up to the data *analyst* to specify. In contrast, in Overlook, the data *curator* specifies the bin boundaries, and can therefore choose a set of boundaries that is appropriate for the dataset and provides good visual utility to analysts.

Importantly, Overlook never fully materializes a quantized version of the dataset. Instead, the quantization policy is expressed *implicitly* as a function that maps data points their quantized versions.

3. Definitions and Data Model

We consider a tabular dataset $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_d$, which consists of n rows and d columns or attributes. This dataset could be the result of a join on multiple tables or a materialized view.[3] Each column i takes values in the domain $\mathcal{A}_i$, which must be finite and public (which can be achieved implicitly by applying the privacy policy to each column at query time).

Our goal is to answer queries from some set $\mathcal{Q}$, where each $q \in \mathcal{Q}$ is a function $q : \mathcal{A}^n \to \mathcal{Y}$. A mechanism for answering queries from $\mathcal{Q}$ is a randomized algorithm $\mathcal{M} : \mathcal{A}^n \times \mathcal{Q} \to \mathcal{Y}$. Following [14], a mechanism $\mathcal{M}$ is ε-differentially private if for any two databases X, X' which differ in a single row,

$$\forall Y \subseteq \mathcal{Y}, \Pr[\mathcal{M}(X, q) \in Y] \leq e^{\varepsilon} \Pr[\mathcal{M}(X', q) \in Y].$$

Intuitively, this means that adding, removing, or changing any one row of the dataset will not change the probability of an event under the differentially-private mechanism by more than a pre-specified multiplicative factor, e^{ε}. An extensive line of work has explored the construction of mechanisms that achieve this guarantee [16, 48]. The most common mechanisms add random noise to the raw result according to the Laplace distribution. In a histogram visualization, this manifests as random perturbations to the counts for each bar. Our challenge is then to choose a mechanism that gives good visual utility to the user despite these random perturbations (§3.2) and implement it efficiently (§4).

Importantly, throughout this section we assume that the mechanism operates over data which has *already* been quantized according to the privacy policy, and therefore belongs to a finite and *public* domain $\mathcal{A}$. In practice, this quantization happens on demand, at query time; the quantized dataset is not materialized.

3.1. Query model.

Query model. Overlook primarily supports one- and two-dimensional histogram queries.[4]

The basic building block of a histogram query is computing the count of the elements that belong to a bucket; a bucket is in general a k-dimensional rectangle.

For example, a 1-dimensional histogram query with ℓ buckets specifies a column i and a set of $\ell + 1$ bucket boundaries $h_0 < h_1 < \ldots < h_\ell$. The query returns a vector of ℓ counts, one for each interval $[h_i, h_{i+1})$.

[3] We note that a line of prior work [41, 6, 39, 27, 31] considers limiting the sensitivity of joins in differential privacy. Overlook assumes all released views are materialized and provides only per-row privacy in the materialized view, but, as Overlook operates over a standard database backend, these techniques could be incorporated in the future.

[4] Larger dimensions can be accommodated easily, but histograms and heat maps are most relevant to the visualization setting.

Note that, although the domain $\mathcal{A}_i$ must be finite and public, the bucket boundaries h_i can be any value specified by the user. For example, if $A_i = \{0, 1, 2\}$, the user may query the range $[0.5, 1.5)$. In this case, Overlook would privately return the (noisy) count corresponding to the value 1 (the only value in the quantized domain that falls in the query range). We are careful to note that this count corresponds to the data *after* quantization.

3.2. **Synopsis mechanism.** Certain families of queries permit a special type of mechanism that produces a private summary called a synopsis. Such a mechanism can be decomposed into two stages:

(1) It releases a synopsis $S = \mathcal{M}_\varepsilon(X)$ of the dataset, which is guaranteed to be ε-differentially private, and is independent of the queries $q \in \mathcal{Q}$.
(2) On input $q \in \mathcal{Q}$, it computes an answer $S(q)$ using only the synopsis. Since the answer is computed by post-processing the synopsis, and post-processing does not leak privacy [14], the answer is differentially private. Further, there is no limit to the number of queries that a user can submit.

Given a column of a dataset over a finite, enumerable data domain $\mathcal{A}_i$ of size m, one can build a histogram of m buckets containing the count of each element in the domain. Making this histogram private naïvely requires adding Laplace noise with scale $\text{Lap}(1/\varepsilon)$ to each of m buckets. Answering an interval query of size t then adds t independent random Laplace variables to the result. The error of such a mechanism scales as $\sqrt{t}$ [9].

However, adding this much noise to each query is suboptimal. Instead, Overlook uses a mechanism called the *hierarchical histogram* [24, 9], also referred to as $\mathcal{H}_b$ in the literature. The error of this mechanism scales as $\log_b^2(t)$ rather than $\sqrt{t}$. At a high level, the hierarchical histogram builds a *tree* such that nodes higher in the tree correspond to progressively larger contiguous intervals in the domain. Each internal node of the tree corresponds to an interval of the histogram that is the union of its b children, and the mechanism adds noise with scale $\text{Lap}(\log_b(m)/\varepsilon)$ to each internal node. For such a tree with branching factor b, an arbitrary interval of size t can be computed by taking the union of only $(b-1)\log_b(t)$ nodes: the number of noise variables now scales logarithmically, rather than linearly, in the interval size. Figure 4 shows such a tree with branching factor 2 (also called a "dyadic" tree).

A multidimensional rectangle query can be computed by taking the Cartesian product of its decomposition in each axis, as illustrated in Figure 5, for a heatmap over columns i, j.

3.3. **Discussion.**

3.3.1. *Why use hierarchical histograms.* Hierarchical histograms are just one of many mechanisms that have been proposed in the literature (see the surveys of [24, 44]). We choose to implement hierarchical histograms for two reasons:

(1) *Data-obliviousness.* Hierarchical histogram mechanisms are oblivious to the data. They only need to know the quantization, which is public knowledge in our setting. This makes them particularly suitable for exploratory data analysis. We do not need any expensive processing of the data to compute quantization boundaries, which several data dependent mechanisms require.
(2) *Error guarantees.* A systematic comparative study of various data-dependent and data-indpendent mechanisms was performed by [23]. They found that for large datasets, the $\mathcal{H}_b$ mechanism typically results in less error than any other mechanism.

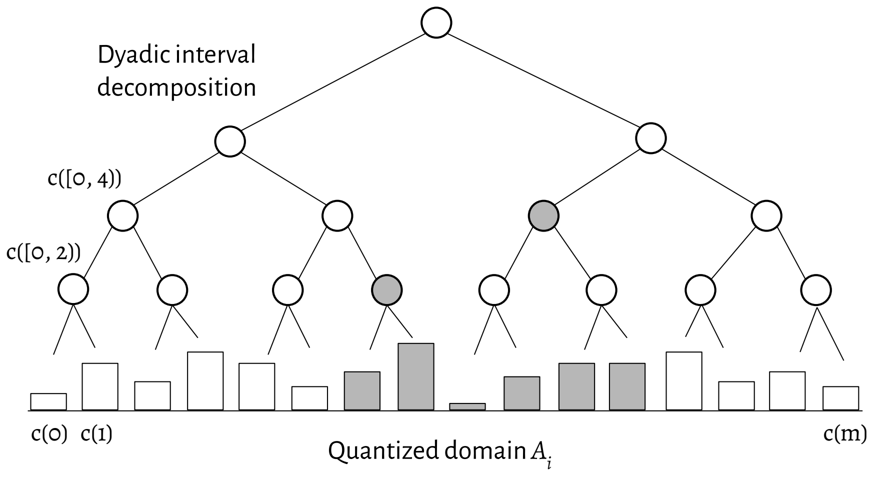


FIGURE 4. Tree used in a hierarchical histogram with $b = 2$. The tree for column i is constructed for the quantized domain A_i. Each internal node in the tree corresponds to the count for a contiguous interval in the domain, and receives independent random Laplace noise. In this example, the grey internal nodes in the tree can be used to compute the count for the grey range in the domain.

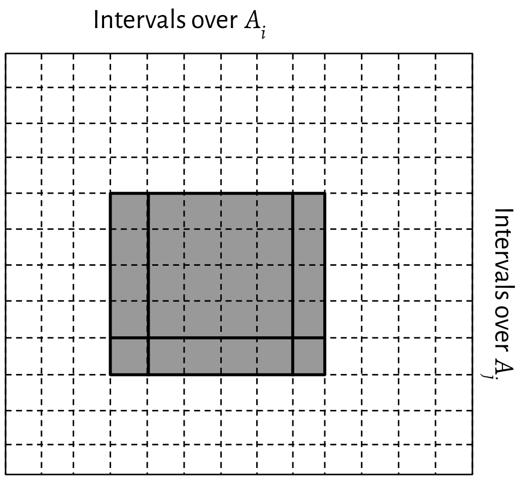


FIGURE 5. Interval decomposition for a 2D histogram (heatmap). The decomposition in two dimensions is simply the Cartesian product of the decompositions in each dimension.

3.3.2. *Optimizing the shape of the tree.* Different tree topologies yield different tradeoffs between accuracy and privacy [44]. The shallower the tree, the less the sensitivity of the resulting synopsis, hence the less noise we need to add per node of the tree. But then the number of nodes we need to sum might be large for some ranges, which means those queries produce noisier results. The tradeoffs between these parameters have been studied extensively in [44]. The best choice of b depends on what type of queries one wishes to optimize for.

3.3.3. *Setting privacy parameters.* The simplest option for the curator is to rely on the basic composition theorem [48, Lemma 2.3] which states that the privacy leakage adds up across mechanisms. Hence, for range queries of dimensionality i the curator might specify a value ε_i, such that $\sum_{i \leq k} \varepsilon_i = \varepsilon$. The curator may then partition each ε_i (uniformly or otherwise) across the d^i mechanisms that answer i-dimensional range queries. Overlook's privacy policy allows the curator to specify the value of ε_S for each set of columns S independently. A curator with greater expertise in differential privacy may take advantage of advanced composition theorems [48] to optimize the choice of ε for a table.

4. Virtual Synopses

Let us start by discussing how a private synopsis is usually constructed and used in differential privacy. An important requirement for releasing a private synopsis is that random noise is added once, when the synopsis is constructed, and must not be resampled on future queries to the synopsis. For the hierarchical histogram mechanism, this requirement naïvely would mean that Overlook would have to store a random sample for every node in the synopsis tree. This incurs a storage overhead that grows linearly in the size of the domain. Further, we would create a different synopsis structure for every histogram, so if there are d columns and we wish to answer all 1 and 2-dimensional histogram queries, this would mean constructing $O(d^2)$ synopsis structures.

Overlook aims to empower the curator to experiment with different privacy settings, and rapidly visualize the effect of these choices on the visualizations rendered for the user. Suppose the curator wishes to experiment with privacy budget allocation for 2-dimensional histograms. Using the naive paradigm as described above would require recomputing $O(d^2)$ synopsis structures every time the budgets change, since the random noise for each histogram would need to be resampled according to the new parameters. This would be costly both in terms of computation time and storage space, and would render it hard for the curator to experiment with a broad set of budgets.

The virtual synopses constructed in Overlook solve this problem. Rather than construct every new synopsis explicitly, they are represented implicitly. When a certain visualization is requested, the virtual synopsis allows us to reconstruct only the relevant part of the synopsis on the fly. This results in a savings in both computation time and storage space, which in turn empowers the curator to experiment with various privacy settings at interactive speeds. The key idea used to construct virtual synopses is to use a cryptographically secure pseudo-random function (PRF) $F : (\mathcal{K}, \mathcal{X}) \to \mathcal{Z}$ to implicitly represent the random noise. This use of random number generators to save space is similar in spirit to the use of such generators in streaming algorithms [3], but in our setting, the use of cryptographically secure generators is essential (see the discussion at the end of the section).

Informally, a PRF guarantees that, given a small random key k in the key space $\mathcal{K}$, $F(k, \cdot)$ will be indistinguishable from a truly random function $R : \mathcal{X} \to \mathcal{Z}$ to a computationally-bounded adversary. (See e.g. [7] for a formal definition.) In our setting, the inputs $\mathcal{X}$

Algorithm 1 Overlook virtual synopsis.

```
Inputs:   range $R$ for which to compute private count
          branching factor $b$
          domain size $m$
          privacy level $\varepsilon$
          column index $i$
          true count $c(R)$
          PRF key $k$
Output:   noisy count $\hat{c}(R)$

Set scale $s = \lceil \log_b(m) \rceil / \varepsilon$
Set $N(R) \leftarrow$ B-adicDecomposition$(R, b)$
Set $\hat{c}(R) = c(R)$
for $v \in N(R)$ do
    $\hat{c}(R) += \ell(F(k, (i, v)), s)$
end for
return $\hat{c}(R)$
```

Algorithm 2 Computing the b-adic decomposition for a range.

```
Inputs:   interval $R = (r_l, r_r)$, $r_l \geq 0, r_r > r_l$
          branching factor $b$
Output:   nodes $N(R) = (v_1, \dots, v_n)$ in tree corresponding to $R$ indexed by (start, interval size)

function B-adicDecomposition(R, b)
    Set $N(R) \leftarrow \{\}$
    Set $L = r_l$
    if $|R| == 0$ then
        return $N(R)$
    end if
    while $L < r_r$ do
        $p_L = -1$;
        if $r_l > 0$ then
            $p_L = \lfloor \log_b(L) \rfloor$
        end if
        $p_s = \lfloor \log_b(r_r - L) \rfloor$
        $p = p_L < 0 \ ? \ p_S : \min(p_L, p_s)$
        nodeSize $= b^p$
        $N(R) = N(R) \cup \{(L, \text{nodeSize})\}$
        $L +=$ nodeSize
    end while
    return $N(R)$
end function
```

to F are nodes in the hierarchical histogram synopsis for a given column, each of which corresponds to a contiguous interval in the underlying domain. F takes as input a node index v, a column index i,[5] and the key k associated with a table, and returns a uniformly distributed random sample $F(k,(i,v))$, which we then transform to a sample from a Laplace variable $\ell(F(k,(i,v)),s)$ with the appropriate scale s.

Using the PRF, we are able to reduce the storage cost of the synopsis from linear in the domain size to a small constant — in fact, only the 32 bytes required to store the key associated with a given table.

We incorporate the PRF into the hierarchical histogram mechanism as follows. A range R can be decomposed into a minimal set $N(R)$ of internal nodes in the synopsis tree, each of which corresponds to a sub-interval of R. For each $v \in N(R)$, we can use the PRF to compute $\ell(F(k,(i,v)),s)$, the random noise corresponding to that interval. Then, if the true count in the interval is $c(R)$, we can compute the private count for the interval as $\hat{c}(R) = c(R) + \sum_{v \in N(R)} \ell(F(k,(i,v)),s)$. (For a domain of size m, the scale s is $\lceil \log_b(m) \rceil/\varepsilon$.)

Algorithm 1 describes the synopsis algorithm in detail. For completeness, we also describe the algorithm for computing a b-adic decomposition of an interval in Algorithm 2.

Because the PRF is ultimately deterministic (though indistinguishable from random), this algorithm satisfies the requirement that queries to the synopsis will not require resampling noise, although we have not explicitly stored any samples.

4.1. **Cryptographic security.** It is important to note that the PRF used to generate random samples must be *cryptographically* secure. If one can somehow reverse-engineer the generator and compute $\ell(k,(i,v))$, then one can subtract it from the end result and obtain the true count $c(v)$. A cryptographic PRF ensures that, even if an adversary knows $\ell(k,(i,v))$ for some v, perhaps because they know $c(v)$ as auxillary information, the adversary still cannot efficiently compute $\ell(k,(j,v'))$ for a new value and column. This additionally requires that the key k associated with a table must be stored securely on the Overlook root node. The importance of securing the randomness in the context of differential privacy along with possible implementational details is discussed in [18].

Section 5.2.1 further describes the implementation of virtual synopses in Overlook.

4.2. **Extension to higher dimensions.** Overlook allows for the computation of histograms in up to 3 dimensions. Higher dimensions than this are not a natural use case for visualization. Here we wish to point out that the complexity of computing a virtual synopsis does scale exponentially with the dimension, in terms of the number of calls to the PRF. Let us assume for simplicity that we have $d = 3$ dimensions, and that in each dimension the domain size is m. Let us fix $B = 2$, so we are computing dyadic decompositions. A histogram query asks for the count in an interval $I_1 \times I_2 \times I_3$ where each I_j is an interval in the domain. We then compute the dyadic decomposition of each interval as $I_j = D_j^1 + \cdots D_j^{k_j}$ where $k \leq \log m$, and then take the product to get

$$\prod_{j-1}^{3} I_j = \prod_{j=1}^{3} (D_j^1 + \cdots D_j^{k_j}) = \sum_{i_1=1}^{k_1} \sum_{i_2=1}^{k_2} \sum_{i_3=1}^{k_3} D_1^{i_1} D_2^{i_2} D_3^{i_3}.$$

This gives a sum of at most $(\log(m))^3$ dyadic rectangles of the form $D_1^{i_1} D_2^{i_2} D_3^{i_3}$, and we query the PRF for a noise variable for each of these intervals.

[5]Multi-dimensional histograms are also assigned unique indexes, which can then be used with the PRF in the same manner; the general PRF for a node in a d-dimensional histogram with index i is $F(k,(i,v_1,\ldots,v_d))$.

Clearly, in d dimensions, this would translate to $(\log(m))^d$ calls. We stress that for Overlook, the natural use cases are $d \leq 2$, while some visualizations do require $d = 3$. The question of whether there are efficient ways to compute virtual synopses in higher dimensions which do not suffer an exponential dependence in d is an interesting open problem.

5. Implementation

We have implemented Overlook by extending the open-source Hillview [8] visualization system. However, to make the case that a system like Overlook could be implemented as an agent between any suitably powerful UI and a generic database, we have modified Hillview to also use a root node that generates SQL queries for a MySQL database for storing the data. We then have added the Overlook differential privacy layer on top of both these back-ends: the Hillview in-memory database and MySQL. For both cases the adaptations required were minimal; the MySQL engine is completely unmodified. We describe them in the following sections.

In this section, we describe implementation details for the UI (§ 5.1), privacy interposition layer (§ 5.2), and Hillview and MySQL backends (§ 5.3).

5.1. **User interface.**

For data interaction and presentation we reuse the UI of Hillview [1]. This UI is written in the TypeScript programming language [5] and runs in any modern web browser. To support differentially private visualizations we had to modify a few hundred lines of code. The most significant changes are (1) the data presentation of uncertainty (confidence intervals) that is inherent in differentially private results, and (2) the curator mode, which enables the curator to edit the privacy parameters interactively. We believe that displaying the confidence intervals significantly enhances the usefulness of a visualization tool.

5.2. **Privacy layer.**

The privacy layer of Overlook is implemented in Java. It is sandwiched between the web server layer and the query generation and execution layer. When a new dataset is opened, the privacy layer checks for the existence of related privacy metadata to decide whether a data source should be treated as differentially private.

5.2.1. *PRFs for virtual synopses.*

The virtual synopsis described in Section 4 uses a PRF to generate noise. In practice, we use AES-256 as the PRF. The root node stores one AES key *per table*, which ensures that no two tables are released using the same PRF. In the same vein, columns and pairs of columns are labeled with unique and immutable IDs so that no two synopses within a table share random samples.

On receiving an interval query $[h_i, h_j)$, the root computes unique IDs of the nodes in the synopsis corresponding to this interval. To generate a new Laplace sample for an interval, the root uses the interval ID and column ID as input to AES to generate random bits, which can then be transformed into a Laplace sample using standard methods for converting bits to doubles and then inverting the Laplace CDF.[6]

[6]We note that we do not currently implement the snapping mechanism described in [38], but this is not fundamental to our system design, and can be incorporated in the future.

5.2.2. *Confidence intervals.* To approximate the α-confidence interval for a sum of Laplace variables, we sample the corresponding distribution and return the $1-\alpha$ percentile value. Naïvely this operation would be performed for every bucket on every histogram query. However, we observe that our synopsis guarantees that every interval will be the sum of at most $\log^d n$ random variables (for a histogram of dimension d, where n is the size of the domain). Therefore, the confidence intervals once computed can be stored in a cache of size at most $\log^d n$. Moreover, the confidence intervals are added as a postprocessing step independent of the raw data, and therefore need not be computed securely; the cache can be shared across columns and tables.

5.2.3. *Privacy policies.* Overlook stores privacy policies in JSON format at the root node. Any user or curator can query the privacy policy, but only the curator is allowed to modify it (before the data is exposed publicly). The current UI allows the curator to edit the JSON file directly.

5.2.4. *Query rewriting.* The query rewriting layer receives queries from the UI and rewrites them to operate on quantized data. We give a concrete example about such query rewriting in Section 5, where we describe the implementation of Overlook using a traditional SQL database. Recall that the quantization parameters for a column are established by the data curator. The quantization information describes a range of intervals for the data values; data that falls outside all the quantization intervals is treated as if it belongs to the same bucket as NULL values. Data out of range could be handled in multiple ways, by allocating essentially different histogram buckets for data that is smaller than the lowest quantization boundary, larger than the largest boundary, or NULL. Our UI handles all 3 categories of data in the same way, but this is just one of the possible choices – each with slightly different utility and privacy costs.

5.2.5. *Adding noise to results.* When the root receives the complete counts for the base histogram, it queries the virtual synopsis for the noise to add to each bucket and adds this noise to the histogram before returning it to the UI. The UI will display the confidence interval centered around the noisy result. The confidence interval around the noisy result may include negative values; in that case the negative values are not displayed, effectively truncating the confidence interval (how to visualize it exactly is up to the UI and the specific visualization). Since it is publicly known that count queries cannot be negative this truncation respects the semantics of a confidence interval and does not violate the privacy constraints.

5.3. **Backends.** Overlook operations can be adapted to use any back-end that supports a rich enough query language to compute standard histograms. We demonstrate this by describing how it operates over two different back-ends: the Hillview back-end, and one that operates on top of MySQL. We are interested in highlighting the additional effort required for adding privacy on top of an existing SQL-based query engine. In this section we describe how this is done for the case of histogram queries.

5.3.1. *Hillview backend.* Hillview is a MapReduce [13]-like distributed query engine that implements *vizketches* – mergeable sketches for visualization. (Histograms are a particular kind of vizketch.) Hillview implements a number of data-parallel aggregation tasks suitable for visualization, including those used in Overlook. Hillview is described in additional detail in [8].

The only change required to support privacy-related processing is to adapt all the existing sketches to first quantize the columns that they operate on according to the appropriate privacy policy. No other changes were required in the backend. This change amounts to less than 10 lines of code in each sketch.

5.3.2. *MySQL backend.* Overlook can interface with unmodified, existing database backends; we have implemented one such backend in MySQL. In this section, we describe some of the queries implemented in order to support the Overlook UI.

Numeric histograms. Consider a user request to display a (non-private) histogram of the data in a column `C` as a histogram with `b` buckets. Let us assume first that `C` is a numeric column in table `t`. This kind of visualization is executed using SQL in two stages: (1) the range of the data in the column is computed, and (2) the histogram is built. To obtain the range of the data we generate the following query:

```sql
SELECT min(C), max(C), count(*), count(C)
FROM t
```

This query computes the minimum and maximum values in column `C`, and also the number of non-null elements and the number of total elements.

The UI receives these parameters and decides on a range `l−r` of data and on a number of buckets `b` to display (in some cases the UI does not need to issue any other query, for example when all elements are `NULL`, or when `l=r`). The query to compute a histogram is written as:

```sql
SELECT bucket, COUNT(bucket) FROM (
  SELECT CAST(FLOOR((C - l) * scale)
     AS UNSIGNED) AS bucket
  FROM t
  WHERE C between l AND r)
GROUP BY bucket
```

`scale=b/(r-l)` is computed statically before the query is generated.

Quantized data view. Since all private queries operate over quantized data, one option is to pre-compute and materialize a view where all columns are quantized using the curator-specified quantization intervals. Such a query can be generated automatically by the system once the privacy policy has been set. For example, to create a view `QV` of a table with a single numeric column `C` with equal-sized quantization intervals of size `g` between `qmin` and `qmax` one can issue the following query:

```sql
CREATE view QV as
  (SELECT qmin + FLOOR((C-qmin)/g)*g AS C
   FROM t WHERE C between qmin AND qmax)
```

Private numeric histograms. For the case of a private numeric column the general flow is very much as described in the previous section, but with three changes: (1) the data range is obtained directly from the privacy policy associated to a column, (2) the query is executed over the quantized view, and (2) after the histogram is computed noise is added to each bucket. Let us assume that we are quantizing the data to be within the range `qmin` and `qmax` with a granularity `g`.

The complete query that is executed is:

```
-- compute histogram
SELECT bucket, COUNT(bucket) FROM (
  -- compute buckets
  SELECT CAST(FLOOR((C - l) * scale)
     AS UNSIGNED) AS bucket
  FROM QV -- quantized view
  WHERE C between l AND r)
GROUP BY bucket
```

String histograms. Computing (non-private) histograms over a categorical column is a bit more involved because the UI never displays a large number of histogram buckets (more than can be shown on the screen). The first step in computing a histogram over a string column involves computing a set of *distinct quantiles* over the column. For example, if the screen can accommodate 50 columns, then the UI will first issue a query to sort the distinct values in the column and extract 50 equi-distant values from the sorted set. These 50 values will be used as histogram bucket boundaries. If the column has fewer than 50 distinct values then all values will be used as distinct bucket boundaries.

```
SELECT DISTINCT BINARY C AS C FROM t
ORDER BY BINARY C
```

(One has to be careful with the sorting and comparison order: these have to be consistent between the code that computes the buckets and the database code that performs comparisons and sorting. In our case we had to prevent MySQL from doing default case-insensitive string comparisons in order to obtain consistent results — this is why we used the `BINARY` keyword. We will omit it from the subsequent queries.)

To compute a histogram quickly over a set of explicit string buckets the Java code generates an explicit binary search tree using nested SQL `IF` expressions. For example, to build a histogram with buckets separated by strings 'A', 'G', 'M', and 'Z' it generates the following query:

```
SELECT bucket, count(bucket)
FROM (
  SELECT (IF(C<'G',0,IF(C<'M',1,2))) AS bucket
  FROM t
  WHERE C BETWEEN 'A' AND 'Z')
GROUP BY bucket
```

Private string histograms. Finally, computing private histograms requires modifying the query for string histograms in a way similar to numeric private histograms, by quantizing the data in the column first. A quantization policy for a string column is given essentially by a sorted list of strings. The quantization query also makes use of a binary search tree. Let's assume that our quantization boundaries are 'A', 'F', 'N', 'O' and 'Z'. The quantization query is:

```
CREATE view QV as
  (SELECT IF(C<'N', IF(C<'F', 'A', 'F'),
                 IF(C<'O', 'N', 'O')) AS C
   FROM t
   WHERE C BETWEEN 'A' AND 'Z')
```

The query to compute a histogram over a quantized view is the composition of these two queries:

```
SELECT bucket, count(bucket)
FROM (
  SELECT (IF(C<'G',0,IF(C<'M',1,2))) AS bucket
  FROM QV -- query the quantized view
  WHERE C BETWEEN 'A' AND 'Z')
GROUP BY bucket
```

6. Experience

We have not carried any standardized user experience tests, either on Hillview or Overlook. In this section we describe the developers' personal experience with the system.

Since we have built Overlook on top of the UI of an existing visualization tool, we can make a direct comparison of the user experience for traditional and differentially-private visualization. In this section, we describe some notable differences between these user experiences.

6.1. **Browsing individual data items.**

The most conspicuous difference is that many operations that are natural in a normal visualization are unavailable when doing differentially-private visualization. For example, enumerating the rows of a table is something that cannot be done in a differentially-private way. Traditional visualization systems can be used for two purposes: detecting trends and identifying outliers. A differentially-private visualization system can only be used for the first purpose: differential privacy masks rare events.

6.2. **Displaying uncertain values.**

A second difference is that all counts that are displayed in a differentially-private visualization are noisy. This can be interpreted as displaying a value with uncertain range. Although there is substantial work on the visualization of uncertain values, from our experience the interpretation of confidence intervals requires a sophisticated understanding from users. While confidence intervals are a useful tool to visualize uncertainty, they do not prevent spurious high counts that users might confuse for signal. This phenomenon has already been observed by [52].

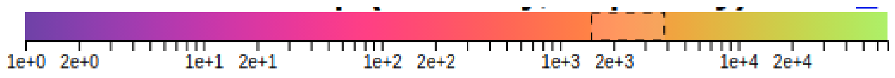


FIGURE 6. An example of a confidence interval (dotted box) overlaid on a heat map legend. When a user hovers over a heat map cell, Overlook highlights the confidence interval corresponding to the cell on the legend.

6.3. **Uncertainty in heat maps.** While uncertainty for histograms can intuitively be presented as a range around each count, it is less clear how uncertainty should be displayed in a heat map. We prototyped multiple possible solutions to this problem.

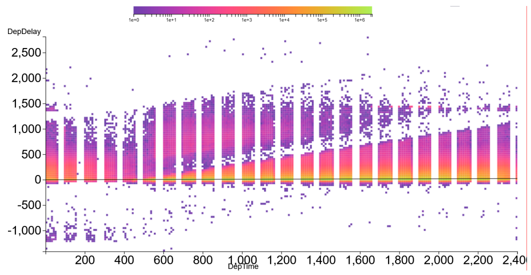
Figure 6 shows a heat map legend in Overlook that displays a confidence interval. When the user hovers over a cell, Overlook highlights the confidence interval in the legend.

In addition, however, we would like to visually convey the confidence in each cell on the chart itself. One way to achieve this, shown in Figure 7c, is to suppress any values whose count is smaller than a multiplicative factor of its confidence interval. A a noisy value $\hat{c}$ with confidence w is considered "low confidence" if $\hat{c} - t \cdot w < 0$, for a threshold parameter t. By default $t = 2$, but we provide an UI to the (untrusted) user to change the value of the threshold t interactively for each plot. Notice that changing t has no privacy costs, it is only changing the way data is displayed on the screen. Figure 7b shows a plot produced with a threshold of $t = 0$, while Figure 7c shows the same data with a threshold of $t = 2$.

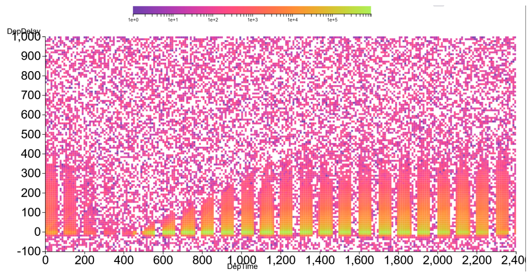
Another idea is to use the *whiteness* of the image to convey uncertainty. In Figure 7d, we illustrate such a plot. The raw color scale is used to convey the count in each cell, and the whiteness provides an additional dimension that can be used to convey the amount of certainty a user should have in the visualization.

6.4. **Quantization intervals.** The quantization intervals, especially for categorical data, have a huge impact on the information that is conveyed to the user. As an example, we show in Figure 9 three histograms of the exact same dataset on the column "cities" with different quantization intervals. The first histogram has on the X axis only the cities that actually appear in the dataset, sorted alphabetically. The second one is quantized on the first letter of a city's name, having exactly 26 buckets. The third one is quantized on the first 2 letters of the city's name. The third one allows the user to zoom-in further and explore the distribution of data for each letter pair; the additional structure is visible in the CDF which is more fine-grained (it has 26^2 steps instead of just 26).

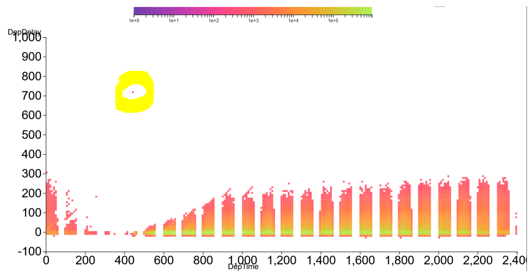
6.5. **Query resolution.** The amount of noise added to a histogram/heatmap bucket R on columns S depends on two primary factors: the extent of the bucket (the set of quantization intervals that fall in R), and the ε_S privacy budget allocated for the set S of columns. The noise does not depend on the actual data distribution; however, the *relative* noise added does depend on the number of data items that falls into the bucket R. So there is a trade-off between the resolution of the query and its precision: if we make buckets smaller, we can potentially see more detail in the data, but the relative noise will be higher. If we make buckets larger we lose the resolution but we gain precision. There is no obvious choice in this trade-off, since it depends very much on the data distribution. This is a trade-off that the data curator can explore and to some degree control by choosing the ε_S and the quantization intervals for each column.



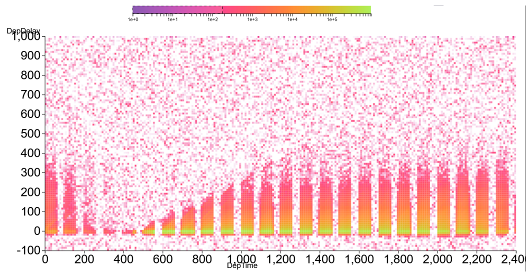
(A) Raw data



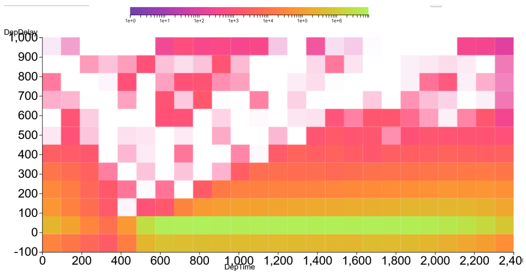
(B) Private heatmap



(C) High confidence values only



(D) Color saturation depicts confidence



(E) Coarse-grain bins

FIGURE 7. Various ways to convey uncertainty. The color scale of the heat map conveys the *value* in each cell. (7a) Raw heatmap with no noise. Y axis range is -1500 – 2800. (7b) Private heatmap with noise and no color adjustment. Notice that the min-max range is different for the Y axis, due to the curator having specified the range -100 – 1000. (7c) Private heatmap; only bins where counts are higher than 2 times the confidence interval of the added noise are displayed. We have highlighted with a yellow circle a lone noisy dot that is displayed. (7d) Whitening added to private heatmap. The counts that have low confidence are desaturated. (7e) Whitening added to private heatmap computed on coarse-grained bins. Each bin now has a larger count relative to the standard deviation of the data, so less whitening is applied.

6.6. **Outliers or sentinel values.** In one database we have encountered a date column which was using a value year of 9999 to indicate that an event has not happened yet. Overlook in general displays counts for `NULL` values separately from the hierarchical histogram synopsis, as `NULL` will not be part of any data range. In contrast, this sentinel value would be naïvely

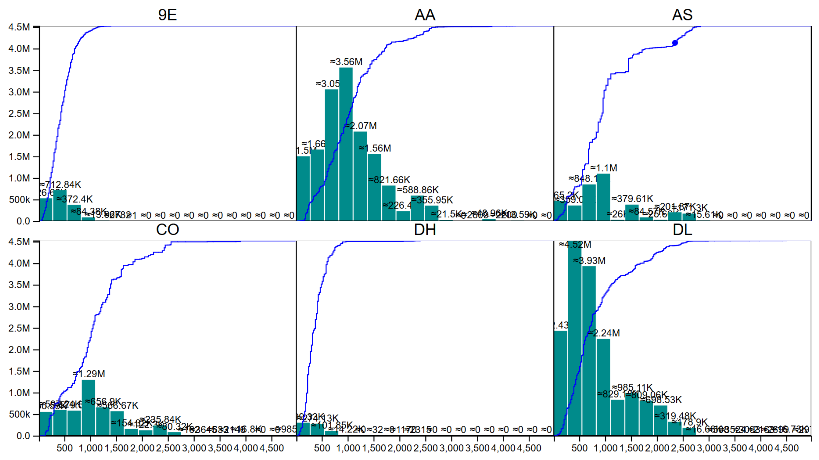


FIGURE 8. A Trellis plot of histograms shows multiple independent histograms, each computed for a range of values in a second column.

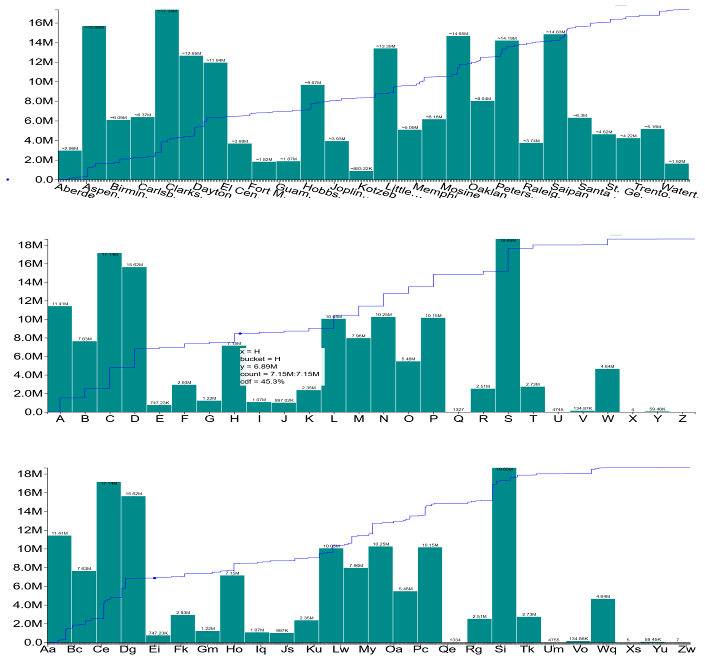


FIGURE 9. Histogram with up to 26 buckets of a set of cities quantized in different ways: (upper) public histogram with adaptively chosen bin boundaries; (middle) quantization fixed to the first letter; (lower) quantization fixed to first two-letters. The shape of the histogram and CDF curve change according to the choice of quantization bins.

included in the displayed histogram if the specified public date range included values up to 9999.

7. Evaluation

In this section, we evaluate the design decisions made in Overlook to support our claims that the system:

- allows data curators to quickly explore parameter settings for synopses before data release (§ 7.2),
- implements a synopsis that provides accuracy comparable to state-of-the-art methods (§ 7.3),
- achieves significantly lower storage cost than the synopses implemented in prior systems through the use of a pseudorandom function (§ 7.4), and
- retains the scaling properties of the underlying distributed system with low performance overhead from privacy (§ 7.5).

In addition, we demonstrate that for large datasets, in the visualization setting, the error induced by differential privacy can be *smaller than a pixel* on the screen – so that, with high probability, the user loses no utility compared to the raw visualization (§ 7.6).

7.1. **Evaluation setup.** Local experiments were run on a machine with 16 GB of memory and 4 cores using an Intel i7 processor. Cloud experiments were run on an Amazon EC2 cluster of 15 machines with 8 GB of RAM and 2 cores each. The Hillview backend uses Java 8.

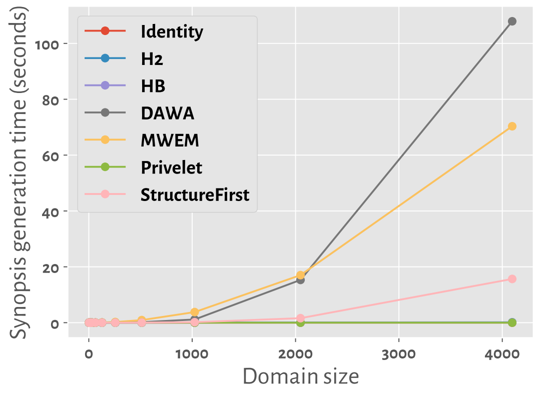
7.2. **Synopsis generation overhead.** One benefit of Overlook is that it allows the data curator to quickly explore privacy parameters for the data before it is released. In particular, the data curator might change the data quantization or privacy budget ε for any column. Generating example visualizations with the new parameters then requires recomputing the underlying synopsis.

We use DPBench [23] to evaluate the time required to generate a synopsis with the hierarchical histogram mechanism against the time required for comparable synopses. We stress that these times are not trivially comparable as DPBench is primarily an accuracy benchmark that is not optimized for performance.

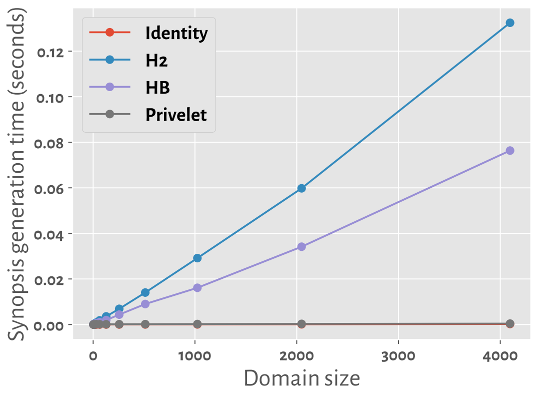
We evaluate each method on a one-dimensional all-zeros dataset of increasing size, on a workload of all intervals (the workload that Overlook targets). These times do *not* include the additional time required to compute the base histogram of counts over which the synopses are computed. We evaluate seven mechanisms in the literature: the baseline "identity" mechanism [14], the binary hierarchical histogram [9, 24], the hierarchical histogram with adaptive branching [24], DAWA [32], MWEM [22], Privelet [49], and StructureFirst [51].

Figure 10 shows the results of the benchmark. Figure 10a shows that MWEM and DAWA are by far the most expensive algorithms, followed by StructureFirst. The remaining algorithms run in under one second, so we plot these separately in Figure 10b. While the time required for the hierarchical mechanisms scales linearly in the data size, they are still considerably less expensive to compute than more complicated workload-aware synopses.

In fact, Overlook itself does not instantiate the synopsis and computes the noisy values on the fly, so the synopsis adds no precomputation overhead but does add some overhead at *query* time. We benchmark Overlook query overhead in Section 7.5.



(A) Time required to generate synopses as the domain size increases.



(B) Generation time for faster synopses.

FIGURE 10. Time required to generate synopses using various mechanisms, benchmarked using DPBench. MWEM and DAWA dominate in the first plot; the second plot shows that generating hierarchical histograms scales in the domain size, when not using Overlook's PRF-based construction.

7.3. **Synopsis accuracy.** Overlook uses hierarchical histograms as the underlying synopsis. The primary benefit of the hierarchical histogram is that the error scales logarithmically, rather than linearly, in the size of the underlying dataset. However, more complex optimization procedures [35, 32] may yield even better accuracy.

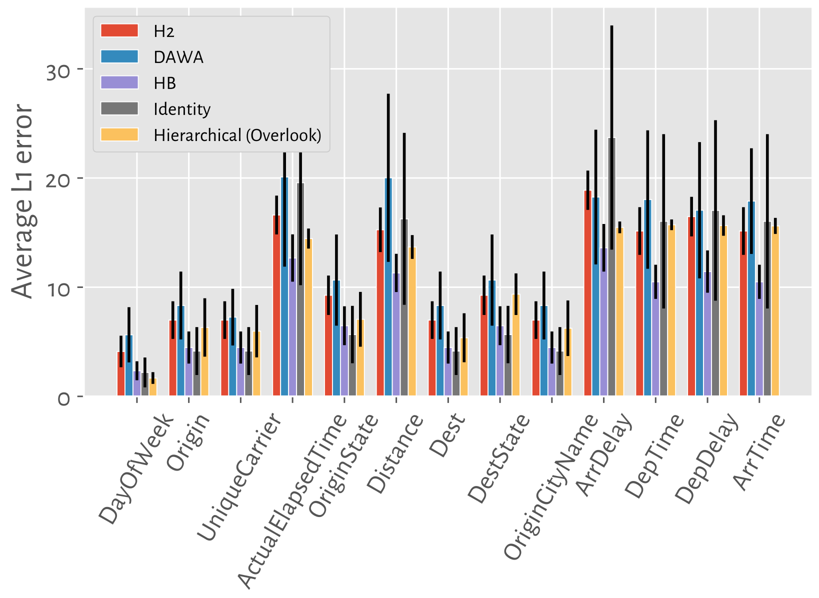
In this section, we demonstrate empirically that Overlook's histogram mechanism achieves utility comparable to that of state-of-the-art synopses. These results are supported by prior work [44, 42, 24] that also investigates the empirical accuracy of hierarchical histograms. More complex methods work well for skewed or restricted query workloads, but Overlook benefits from simple mechanisms because it aims to support a very general set of range queries.

We benchmark accuracy on a dataset of 20 years of U.S. flights [40]. This dataset contains both numeric and categorical columns over a range of data distributions and domain sizes (varying from 7 to over 4000). Figure 11 shows results for histograms on all columns and heat maps on a selection of column pairs. For each bar, we sample 5000 random intervals or rectangles and compute the ℓ_1 distance between the vector of true counts for all samples and the vector of noisy counts returned by the mechanism.

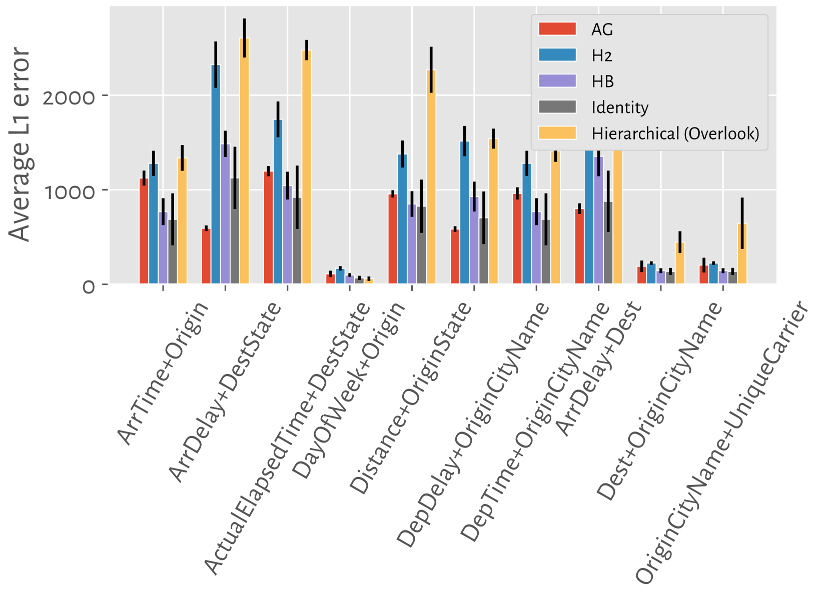
The key takeaway from these figures is that the hierarchical histogram mechanism has comparable accuracy to mechanisms that perform more complex, workload-specific optimization on the random-intervals workload. As noted in [44], the benefits of this mechanism decrease as the dimension increases. Adaptively choosing which mechanism to use for a given visualization may be a direction for future work.

7.4. **Synopsis memory overhead.** Overlook's synopsis mechanism uses only 32 bytes of memory, required to store the AES secret key.

The synopsis mechanisms implemented by DPBench are *consistent* mechanisms. These take as input a histogram over the elements of the domain and output a synthetic histogram



(A) Histogram accuracy. Most mechanisms perform comparably on this dataset. The X axis is the column whose histogram is computed.



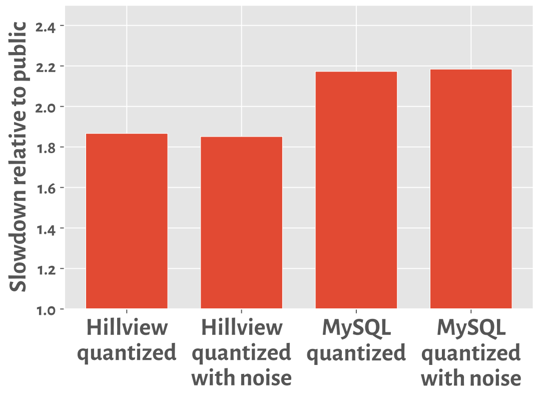
(B) Heatmap accuracy. The baseline identity mechanism outperforms all others; the hierarchical mechanism nevertheless achieves reasonable accuracy. The X axis is a pair of columns whose heatmap is computed.

FIGURE 11. ℓ_1 error of Overlook mechanism on the U.S. flights dataset on 5000 randomly-sampled queries per column or pair of columns.

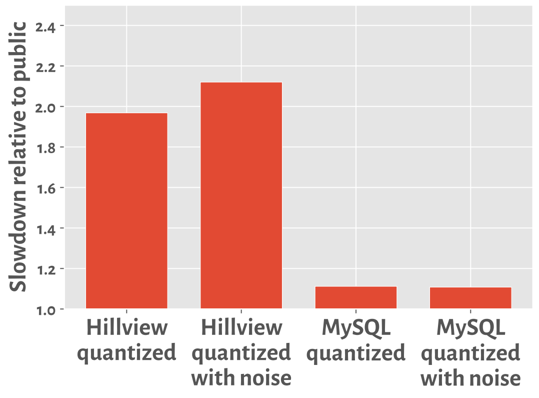
as the synopsis over which all subsequent queries are run. The size of the synopsis is therefore proportional to the size of the data domain for a histogram, and grows exponentially in the number of dimensions (columns). (In particular, if the data is small or especially sparse in the domain, the size may be considerably larger than necessary to represent the data.)

We note, that these mechanisms may require a considerably larger amount of memory at *computation* time. For example, DAWA in two dimensions requires instantiating a matrix representation of the workload [32]. For the workload that Overlook supports (the all-queries workload), this requires a matrix of size n^3. For such a workload, a relatively small domain with 1000 quantization intervals would require at least 8 gigabytes of memory simply to compute the synopsis.

7.5. **Overlook performance.** In this section, we benchmark the performance of Overlook and demonstrate that adding differential privacy does not substantially slow down the system or change its underlying scaling properties. In particular, the overall slowdown from privacy is no greater than 2.2×.



(A) Histogram slowdown.

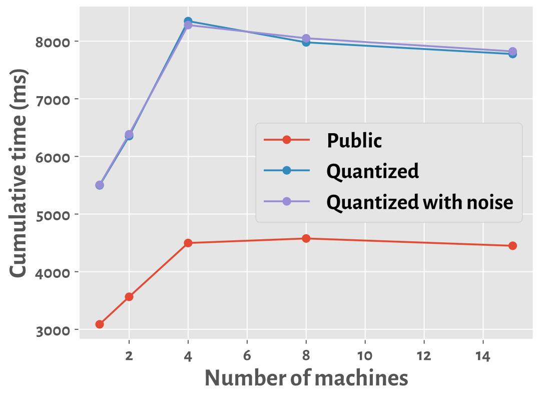


(B) Heatmap slowdown.

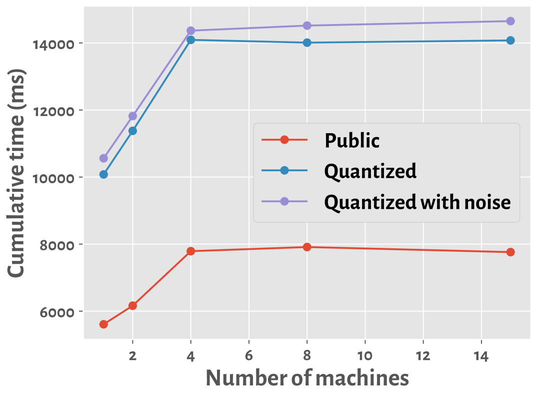
FIGURE 12. Slowdown relative to raw (non-private) databases for histograms and heat maps. In all cases, privacy adds at most a 2.2× performance penalty.

7.5.1. *Slowdown relative to public data.* We first evaluate how much differential privacy causes queries to slow down relative to queries on public data. In order to understand the slowdown, we make two measurements for each backend: first, the time required to quantize the dataset, and second, the time required to answer a quantized histogram query with noise added.

Figure 12 shows the average slowdown when plotting histograms and heat maps on the U.S. flights dataset using both the Hillview and MySQL backends. The slowdown is below 2.5× for all configurations. In all cases, the majority of the slowdown is a result of the quantization step. This is intuitive: where each data point would initially have required one operation to add it to the appropriate bucket, quantization adds an additional operation to round the point to its nearest value in the quantization domain.



(A) Histogram scaling.



(B) Heat map scaling.

FIGURE 13. Average time to generate histograms for columns in the flights dataset as the number of machines grows. The data size grows with the number of machines, so the runtime remains constant.

7.5.2. *Scaling.* The Overlook frontend can be used with any SQL backend. However, the Hillview distributed backend is powerful as it retains Hillview's ability to scale to large datasets.

We evaluate scaling using clusters of 1, 2, 4, 8, and 15 Amazon EC2 machines. The total dataset size is 58.2 GB, split equally among the machines in the cluster. For linear scaling we expect the time required for each query to be roughly the same regardless of the number of machines. We measure time to compute charts once the data is already in memory (excluding I/O time).

In Figure 13, we show our measurements that evaluate the time breakdown for computing histograms over the U.S. flights dataset. Each point corresponds to the total time required to compute a histogram or heat map for every column or pair of columns. The overhead of privacy is the same roughly $2\times$ overhead as in Figure 12, but privacy does not change the scaling behavior of the system at all, as expected.

7.6. **Visual error.**

At large enough data sizes, the error induced by differential privacy can be *smaller* than the pixel-level rounding error induced by the screen resolution. In particular, in a 1-dimensional histogram, Overlook rescales the y-axis to the maximum displayed value in order to make use of all of the available vertical pixels.

Given p vertical pixels and a maximum displayed value of y, each pixel represents a count of y/p. Hence, a confidence interval of size less than y/p will be smaller than a pixel on the screen. Assuming the case of a single Laplace random variable added to each bucket (i.e. the "identity" mechanism), we ask what value of ε would suffice to achieve this level of error.

The inverse of the Lap$(1/\varepsilon)$ distribution is given by

$$F^{-1}(x) = -(1/\varepsilon)\,\mathrm{sgn}(p-0.5)\ln(1-2|x-0.5|).$$

Then we would like $F^{-1}(0.95) < y/p$ for a confidence level of $\alpha = 0.95$. In this case, we arrive at an approximate solution of $\varepsilon > 2.303p/y$. In other words, if the maximum count is approximately **2.3 times** the number of vertical pixels, a privacy level of $\varepsilon = 1$ will likely result in *no visible difference* from the raw data.

8. Related Work

8.1. **Differentially private database management systems.** A number of prior systems make differential privacy available through a SQL-like database API. PINQ [37] implements a subset of SQL as well as a prototype visualization system with incremental ε-budgeting. PINQ additionally pointed out that joins have potentially unbounded sensitivity, and several later systems [41, 6, 39, 27] propose methods for mitigating this issue.

Other work considers additional variants on SQL-like programming frameworks that allow developers to easily express differentially private queries. Airavat [45] allows users to run custom MapReduce [13] queries on sensitive data by enforcing differential privacy on the queries. Ektelo [53] exposes a number of higher-level operators as a programming framework for differentially private mechanisms. PrivateSQL [31] uses synopsis-based mechanisms rather than incremental budgeting to release dataset views. PrivateSQL introduces the notion of *view sensitivity* as an approach to handle joins. While Overlook does not explicitly target joins, such techniques could be naturally integrated with Overlook, as the join is ultimately materialized as a tabular view of the data. Chorus [29] implements differential privacy directly in SQL. APEx [19] supports adaptively-chosen exploratory queries presented in a SQL-like declarative format.

Most of these prior systems support a broad set of queries written directly in SQL or a SQL-like language. In contrast, Overlook restricts queries to those visualizations enabled by the UI, which enables the release of a flexible and small synopsis.

8.2. **Synopsis-based mechanisms.** A number of DP mechanisms [48] are designed to support all queries in a given class of queries simultaneously. Recent work on synopses include the matrix mechanism [33, 34, 35], HDMM [36] where the workload is given implicitly, wavelet transforms [49], and approaches that incorporate data-dependent partitioning [50, 51, 42, 12]. Overlook primarily relies on a hierarchical histogram [24, 9]. A growing body of work [44, 43, 23] additionally considers data-dependent optimizations that can improve the accuracy of synopses under certain query workloads.

A number of papers [44, 42, 24] have investigated the accuracy of these methods in practice. These papers support our claim that the synopses used in Overlook give usable, and often optimal, accuracy in practice.

The idea of public quantization boundaries, or partitions, has been explored by PINQ [37] and FLEX [27]. Both of these systems leave it to the data analyst, rather than the data curator, to specify the quantization boundaries.

Similar ideas are used to analyze streaming data in [20, 10]. A data aware version of the binary mechanism is presented in [2, 32]. It uses a private partitioning method that smooths regions of similar count.

Another class of work aims to release a synthetic dataset that approximates the empirical distribution of the data [54, 26] to answer unrestricted queries. Typically, because the query class is not restricted, these methods must incur more error per query.

8.3. **Differentially private visualization.** PSI [17] may be the closest system to Overlook; PSI makes ε-budgeting more user-friendly by providing a visual interface for users to interact with and understand the impact of various values of ε. In contrast to Overlook, PSI assumes a per-user, incremental ε budget; additionally, PSI is not targeted toward the data exploration use case.

PINQ [37] provides a case study of a differentially private map visualization as an application of the framework. [11] provides methods to make linear and logistic regression plots differentially private. VisDPT [25] is an interface to view two-dimensional trajectories in a differentially private manner. [52] studies the challenges involved in creating meaningful visualizations under differential privacy.

9. Conclusion

We have presented Overlook, a visualization system for private data that provides interactive latencies both for data curators and data analysts. Overlook's novel virtual synopsis enables it to scale to large data domains while incurring minimal performance and storage overhead over queries to raw data. Overlook can integrate with existing query engines with no intrusive changes. Overlook makes differential privacy accessible, useful, and performant, making it a practical privacy tool for the real world.

References

[1] Hillview: a big data spreadsheet. http://github.com/vmware/hillview. Retrieved January 2021.

[2] G. Acs, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *2012 IEEE 12th International Conference on Data Mining*, pages 1–10, Washington, DC, USA, 2012. IEEE Computer Society, pages 1–10. https://doi.org/10.1109/ICDM.2012.80.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *ACM Symposium on Theory of Computing (STOC)*, pages 20–29, 1996, pages 20–29. https://doi.org/10.1145/237814.237823.

[4] Apple. Differential privacy — technical overview. https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf, 2017.

[5] G. Bierman, M. Abadi, and M. Torgersen. Understanding TypeScript. In R. Jones, editor, *ECOOP 2014 – Object-Oriented Programming*, pages 257–281, Berlin, Heidelberg, 2014. Springer, pages 257–281. https://doi.org/10.1007/978-3-662-44202-9_11.

[6] J. Blocki, A. Blum, A. Datta, and O. Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In *Conference on Innovations in Theoretical Computer Science*, pages 87–96. ACM, 2013, pages 87–96. https://doi.org/10.1145/2422436.2422449.

[7] D. Boneh and V. Shoup. A graduate course in applied cryptography. *Version 0.5*, 2020. URL: https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_5.pdf.

[8] M. Budiu, P. Gopalan, L. Suresh, U. Wieder, H. Kruiger, and M. K. Aguilera. Hillview: A trillion-cell spreadsheet for big data. *Proc. VLDB Endow.*, 12(11):1442–1457, July 2019. https://doi.org/10.14778/3342263.3342279.

[9] 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):26, 2011. https://doi.org/10.1145/2043621.2043626.

[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 (CCS)*, pages 1375–1388, 2017, pages 1375–1388. https://doi.org/10.1145/3133956.3134102.

[11] Y. Chen, A. Machanavajjhala, J. P. Reiter, and A. F. Barrientos. Differentially private regression diagnostics. In *ICDM*, pages 81–90, 2016, pages 81–90. https://doi.org/10.1109/ICDM.2016.0019.

[12] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu. Differentially private spatial decompositions. In *2012 IEEE 28th International Conference on Data Engineering*, pages 20–31. IEEE, 2012, pages 20–31. https://doi.org/10.1109/ICDE.2012.16.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, December 2004. URL: http://labs.google.com/papers/mapreduce.html.

[14] C. Dwork. Differential privacy. *Encyclopedia of Cryptography and Security*, pages 338–340, 2011. URL: https://doi.org/10.1007/978-1-4419-5906-5.

[15] C. Dwork and J. Lei. Differential privacy and robust statistics. In *Forty-first Annual ACM Symposium on Theory of Computing (STOC)*, pages 371–380, 2009, pages 371–380. URL: http://doi.acm.org/10.1145/1536414.1536466, https://doi.org/10.1145/1536414.1536466.

[16] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014. URL: https://doi.org/10.1561/0400000042.

[17] M. Gaboardi, J. Honaker, G. King, K. Nissim, J. Ullman, and S. P. Vadhan. PSI: a private data sharing interface. *CoRR*, abs/1609.04340, 2016. URL: http://arxiv.org/abs/1609.04340.

[18] S. L. Garfinkel and P. Leclerc. Randomness concerns when deploying differential privacy. In *WPES'20: Proceedings of the 19th Workshop on Privacy in the Electronic Society*, pages 73–86, Virtual Event, USA, November 9 2020. ACM, pages 73–86. https://doi.org/10.1145/3411497.3420211.

[19] C. Ge, X. He, I. F. Ilyas, and A. Machanavajjhala. APEx: Accuracy-aware differentially private data exploration. In *2019 International Conference on Management of Data*, pages 177–194, 2019, pages 177–194. https://doi.org/10.1145/3299869.3300092.

[20] S. Ghayyur, Y. Chen, R. Yus, A. Machanavajjhala, M. Hay, G. Miklau, and S. Mehrotra. IoT-detective: Analyzing IoT data under differential privacy. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018*, pages 1725–1728, Houston, TX, June 10-15 2018. pages 1725–1728. https://doi.org/10.1145/3183713.3193571.

[21] Google. Privacy on a beam. https://github.com/google/differential-privacy/tree/main/privacy-on-beam, 2021.

[22] M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 61–70. IEEE, 2010, pages 61–70. https://doi.org/10.1109/FOCS.2010.85.

[23] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using DPBench. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, pages 139–154, June 26 – July 01 2016, pages 139–154. https://doi.org/10.1145/2882903.2882931.

[24] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3(1-2):1021–1032, Sept. 2010. https://doi.org/10.14778/1920841.1920970.

[25] X. He, N. Raval, and A. Machanavajjhala. A demonstration of VisDPT: visual exploration of differentially private trajectories. *Proceedings of the VLDB Endowment*, 9(13):1489–1492, 2016. https://doi.org/10.14778/3007263.3007291.

[26] Z. Huang, R. McKenna, G. Bissias, G. Miklau, M. Hay, and A. Machanavajjhala. PSynDB: accurate and accessible private data generation. *Proceedings of the VLDB Endowment*, 12(12):1918–1921, 2019. https://doi.org/10.14778/3352063.3352099.

[27] N. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for SQL queries. *Proc. VLDB Endow.*, 11(5):526–539, Jan. 2018. https://doi.org/10.1145/3187009.3177733.

[28] N. M. Johnson. *Towards Practical Privacy-Preserving Data Analytics.* PhD thesis, University of California, Berkeley, USA, 2018. URL: http://www.escholarship.org/uc/item/72j4b4n5.

[29] N. M. Johnson, J. P. Near, J. M. Hellerstein, and D. Song. Chorus: Differential privacy via query rewriting. *CoRR*, abs/1809.07750, 2018. URL: http://arxiv.org/abs/1809.07750.

[30] A. Khalid. Google is making its differential privacy tool available to all developers. Engadget, 2019. URL: https://www.engadget.com/2019-09-05-google-is-making-its-differential-privacy-tool-available-to-all.html.

[31] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. PrivateSQL: A differentially private SQL query engine. *PVLDB*, 12(11):1371–1384, 2019. URL: http://www.vldb.org/pvldb/vol12/p1371-kotsogiannis.pdf.

[32] C. Li, M. Hay, G. Miklau, and Y. Wang. A data- and workload-aware query answering algorithm for range queries under differential privacy. *PVLDB*, 7(5):341–352, 2014. https://doi.org/10.14778/2732269.2732271.

[33] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 123–134. ACM, 2010, pages 123–134. https://doi.org/10.1145/1807085.1807104.

[34] C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. *Proceedings of the VLDB Endowment*, 5(6):514–525, 2012. https://doi.org/10.14778/2168651.2168653.

[35] C. Li, G. Miklau, M. Hay, A. McGregor, and V. Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB journal*, 24(6):757–781, 2015. https://doi.org/10.1007/s00778-015-0398-x.

[36] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *Proc. VLDB Endow.*, 11(10):1206–1219, 2018. https://doi.org/10.14778/3231751.3231769.

[37] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *ACM SIGMOD International Conference on Management of Data*, pages 19–30, 2009, pages 19–30. https://doi.org/10.1145/1559845.1559850.

[38] I. Mironov. On significance of the least significant bits for differential privacy. In *ACM Conference on Computer and Communication Security (CCS)*, pages 650–661, 2012, pages 650–661. https://doi.org/10.1145/2382196.2382264.

[39] A. Narayan and A. Haeberlen. DJoin: differentially private join queries over distributed databases. In *Symposium on Operating System Design and Implementation (OSDI)*, pages 149–162, 2012, pages 149–162. URL: https://www.usenix.org/system/files/conference/osdi12/osdi12-final-164.pdf.

[40] U. D. of Transportation. Airline on-time performance data. https://transtats.bts.gov/Tables.asp?DB_ID=120. Retrieved December 2020.

[41] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis. *PVLDB*, 7(8):637–648, 2014. https://doi.org/10.14778/2732296.2732300.

[42] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *29th international conference on data engineering (ICDE)*, pages 757–768, 2013, pages 757–768. https://doi.org/10.1109/ICDE.2013.6544872.

[43] W. Qardaji, W. Yang, and N. Li. PriView: Practical differentially private release of marginal contingency tables. In *SIGMOD International Conference on Management of Data*, pages 1435–1446, 2014, pages 1435–1446. https://doi.org/10.1145/2588555.2588575.

[44] W. H. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 6(14):1954–1965, 2013. https://doi.org/10.14778/2556549.2556576.

[45] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for MapReduce. In *NSDI*, volume 10, pages 297–312, 2010, 10:297–312. URL: https://www.usenix.org/legacy/events/nsdi10/tech/full_papers/roy.pdf.

[46] Tableau Software. Tableau. https://www.tableau.com, Retrieved January 2021.

[47] US Census Bureau. Disclosure avoidance and the 2020 census. https://www.census.gov/about/policies/privacy/statistical_safeguards/disclosure-avoidance-2020-census.html, Retrieved 2019.

[48] S. Vadhan. *The Complexity of Differential Privacy*, pages 347–450. Springer International Publishing, Cham. pages 347–450, 2017. https://doi.org/10.1007/978-3-319-57048-8_7.

[49] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on knowledge and data engineering*, 23(8):1200–1214, 2010. https://doi.org/10.1109/TKDE.2010.247.

[50] Y. Xiao, L. Xiong, L. Fan, S. Goryczka, and H. Li. DPCube: Differentially private histogram release through multidimensional partitioning. *Trans. Data Privacy*, 7(3):195–222, Dec. 2014. https://doi.org/10.5555/2870614.2870615.

[51] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *The VLDB Journal—The International Journal on Very Large Data Bases*, 22(6):797–822, 2013. https://doi.org/10.1007/s00778-013-0309-y.

[52] D. Zhang, M. Hay, G. Miklau, and B. O'Connor. Challenges of visualizing differentially private data. In *Theory and Practice of Differential Privacy*, 2016. URL: https://people.cs.umass.edu/~miklau/assets/pubs/viz/zhang16challenges.pdf.

[53] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau. Ektelo: A framework for defining differentially-private computations. In *2018 International Conference on Management of Data*, SIGMOD '18, pages 115–130, 2018, pages 115–130. https://doi.org/10.1145/3183713.3196921.

[54] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. PrivBayes: Private data release via Bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):1–41, 2017. https://doi.org/10.1145/3134428.