Cache Me If You Can: Accuracy-Aware Inference Engine for Differentially Private Data Exploration

Miti Mazmudar
University of Waterloo
miti.mazmudar@uwaterloo.ca

Thomas Humphries
University of Waterloo
thomas.humphries@uwaterloo.ca

Jiaxiang Liu
University of Waterloo
j632liu@uwaterloo.ca

Matthew Rafuse
University of Waterloo
matthew.rafuse@uwaterloo.ca

Xi He
University of Waterloo
xi.he@uwaterloo.ca

ABSTRACT

Differential privacy (DP) allows data analysts to query databases that contain users' sensitive information while providing a quantifiable privacy guarantee to users. Recent interactive DP systems such as APEx provide accuracy guarantees over the query responses, but fail to support a large number of queries with a limited total privacy budget, as they process incoming queries independently from past queries. We present an interactive, accuracy-aware DP query engine, *CacheDP*, which utilizes a differentially private cache of past responses, to answer the current workload at a lower privacy budget, while meeting strict accuracy guarantees. We integrate complex DP mechanisms with our structured cache, through novel cache-aware DP cost optimization. Our thorough evaluation illustrates that *CacheDP* can accurately answer various workload sequences, while lowering the privacy loss as compared to related work.

PVLDB Reference Format:
Miti Mazmudar, Thomas Humphries, Jiaxiang Liu, Matthew Rafuse, and Xi He. Cache Me If You Can: Accuracy-Aware Inference Engine for Differentially Private Data Exploration. PVLDB, 16(4): 574 - 586, 2022.
doi:10.14778/3574245.3574246

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://gitlab.uwaterloo.ca/m2mazmud/cachedp-public.

1 INTRODUCTION

Organizations often collect large datasets that contain users' sensitive data and permit data analysts to query these datasets for aggregate statistics. However, a curious data analyst may use these query responses to infer a user's record. Differential Privacy (DP) [4, 5] allows organizations to provide a guarantee to their users that the presence or absence of their record in the dataset will only change the distribution of the query response by a small factor, given by the privacy budget. This guarantee is typically achieved by perturbing the query response with noise that is inversely proportional to the privacy budget. Thus, DP systems face an accuracy-privacy trade-off: they should provide accurate query responses, while reducing the privacy budget spent. DP has been deployed at the US Census Bureau [17], Google [29] and Microsoft [3].

Existing DP deployments [1, 3, 14, 17] mainly consider a non-interactive setting, where the analyst provides all queries in advance. Whereas in interactive DP systems [7, 12, 20, 29], data analysts supply queries one at a time. These systems have been difficult to deploy as they often assume an analyst has DP expertise. First, data analysts need to choose an appropriate privacy budget per query. Second, data analysts require each DP noisy query response to meet a specific accuracy criterion, whereas DP systems only seek to minimize the expected error over multiple queries. Ge et al.'s APEx [9] eliminates these two drawbacks, as data analysts need only specify accuracy bounds in the form of an error rate α and a probability of failure β. APEx chooses an appropriate DP mechanism and calibrates the privacy budget spent on each workload, to fulfill the accuracy requirements. However, interactive DP systems may run out of privacy budget for a large number of queries.

We observe that we can further save privacy budget on a given query, by exploiting *past*, related noisy responses, and thereby, we can answer a larger number of queries interactively. The DP post-processing theorem allows arbitrary computations on noisy responses without affecting the DP guarantee. Hay et al. [11] have applied this theorem to enforce consistency constraints among noisy responses to related range queries, thereby improving their accuracy, through *constrained inference*. Peng et al. have proposed caching noisy responses and reusing them to answer future queries in Pioneer [24]. However, their cache is unstructured and only operates with simple DP mechanisms such as the Laplace mechanism.

We design a usable interactive DP query engine, *CacheDP*, with a built-in differentially private cache, to support data analysts in answering data exploration workloads accurately, without requiring them to have any knowledge of DP. Our system is built on top of an existing non-private DBMS and interacts with it through standard SQL queries. *CacheDP* meets the analysts' (α, β) accuracy requirements on each workload, while minimizing the privacy budget spent per workload. We note that a similar reduction in privacy budget could be obtained if an expert analyst planned their queries, however our system removes the need for such planning.

Our contributions address four main challenges in the design of our engine. First, we structure our cache to maximize the possible reuse of noisy responses by DP mechanisms (Section 3). Our cache design fully harnesses the post-processing theorem in the interactive setting, for cached noisy responses. Second, we integrate existing DP mechanisms with our cache, namely Li et al.'s Matrix

Proceedings of the VLDB Endowment, Vol. 16, No. 4 ISSN 2150-8097.
doi:10.14778/3574245.3574246

Mechanism [16] (Section 4), and Koufogiannis et al.'s Relax Privacy mechanism [15] (Section 6). In doing so, we address technical challenges that arise due to the need to maintain accuracy requirements over cached responses while minimizing the privacy budget, and thus, we provide a novel privacy budget cost estimation algorithm.

Third, we extend our cache-aware DP mechanisms with two modules, which further reduce the privacy budget (Section 5). Specifically, we apply DP sensitivity analysis to proactively fill our cache, and we apply constrained inference to increase cache reuse. We note that *CacheDP* internally chooses the DP module with the lowest privacy cost per workload, removing cognitive burden on data analysts. Fourth, we develop the design of our cache to handle queries with multiple attributes efficiently (Section 7).

Finally, we conduct a thorough evaluation of our *CacheDP* against related work (APEx, Pioneer), in terms of privacy budget consumption and performance overheads (Section 8). We find that it consistently spends lower privacy budget as compared to related work, for a variety of workload sequences, while incurring modest performance overheads. Through an ablation study, we deduce that our standard configuration with all DP modules turned on, is optimal for the evaluated workload sequences. Thus, researchers implementing our system need not tinker with our module configurations. **This paper contains several theorems and lemmas; their proofs can be found in the extended version of the paper [18]**.

2 BACKGROUND

We consider a single-table relational schema $\mathcal{R}$ across d attributes: $\mathcal{R}(\mathcal{A}_1, \dots \mathcal{A}_d)$. The domain of an attribute $\mathcal{A}_i$ is given by $dom(\mathcal{A}_i)$ and the full domain of $\mathcal{R}$ is $dom(\mathcal{R}) = dom(\mathcal{A}_1) \times \cdots \times dom(\mathcal{A}_d)$. Each attribute $\mathcal{A}_i$ has a finite domain size $|dom(\mathcal{A}_i)| = n_i$. The full domain has a size of $n = \prod_i n_i$. A database instance D of relation $\mathcal{R}$ is a multiset whose elements are values in $dom(\mathcal{R})$.

A predicate $\phi : dom(\mathcal{R}) \rightarrow \{0, 1\}$ is an indicator function specifying which database rows we are interested in (corresponds to the `WHERE` clause in SQL). A linear or row counting query (RCQ) takes a predicate ϕ and returns the number of tuples in D that satisfy ϕ, i.e., $\phi(D) = \sum_{t \in D} \phi(t)$. This corresponds to querying `SELECT COUNT(*) FROM` D `WHERE` ϕ in SQL. We focus on RCQs for this work as they are primitives that can be used to express histograms, multi-attribute range queries, marginals, and data cubes.

In this work, we express RCQs as a matrix. Consider $dom(\mathcal{R})$ to be an ordered list. We represent a database instance D by a data (column) vector $\mathbb{x}$ of length n, where $\mathbb{x}[i]$ is the count of ith value from $dom(\mathcal{R})$ in D. After constructing $\mathbb{x}$, we represent any RCQ as a length-n vector $\mathbb{w}$ with $\mathbb{w}[i] \in \{0, 1\}$ for $i = 1, \dots, n$. To obtain the ground truth response for a RCQ $\mathbb{w}$, we can simply compute $\mathbb{w} \cdot \mathbb{x}$. Hence, we can represent a workload of ℓ RCQs as an $\ell \times n$ matrix $\mathbb{W}$ and answer this workload by matrix multiplication, as $\mathbb{W}\mathbb{x}$.

When we partition the full domain $dom(\mathcal{R})$ into a set of n' disjoint buckets, the data vector $\mathbb{x}$ and the workload matrix $\mathbb{W}$ over the full domain $dom(\mathcal{R})$ can be mapped to a vector $\mathbf{x}$ of size n' and a matrix $\mathbf{W}$ of size $\ell \times n'$, respectively. We also consider a workload matrix $\mathbf{W}$ as a set of RCQs, and hence applying a set operator over a workload matrix is equivalent to applying this operator over a set of RCQs. For example, $\mathbf{W}' \subseteq \mathbf{W}$ means the set of RCQs in $\mathbf{W}'$ is a subset of the RCQs in $\mathbf{W}$. We follow a differential privacy model with a trusted data curator.

Table 1: Notation

Notation	Description
$\mathbb{x}, \mathbb{w}, \mathbb{W}, \mathbb{A}$	raw data vector, query vector, query workload matrix, strategy matrix over full domain $dom(\mathcal{R})$
$\mathbf{x}, \mathbf{w}, \mathbf{W}, \mathbf{A}$	mapped data vector, query vector, query workload matrix, strategy matrix over a partition of $dom(\mathcal{R})$
α, β	accuracy parameters for $\mathbb{W}$
$\mathcal{B}, B_c, \epsilon$	total budget, consumed budget, workload budget
$\mathbb{A}^*, C_{\mathbb{A}^*}$	global strategy matrix, its cache over $dom(\mathcal{R})$
$b, \tilde{y}$	a scalar noise parameter, a scalar noisy response
$\mathbf{b}$	a vector of noise parameters
$\tilde{\mathbf{y}}, \tilde{\mathbf{z}}$	a vector of noisy responses to the strategy $\mathbf{A}$ or $\mathbf{W}$.
$(\mathbb{a}, b, \tilde{y}, t)$	a cache entry for a strategy query $\mathbb{a} \in \mathbb{A}^*$ stored at timestamp t. See Definition 3.1.
$\mathbf{F}, \mathbf{P}$	free strategy matrix, paid strategy matrix

Definition 2.1 (ϵ-Differential Privacy (DP) [4]). A randomized mechanism $M : \mathcal{D} \rightarrow \mathcal{O}$ satisfies ϵ-DP if for any output sets $O \subseteq \mathcal{O}$, and any neighboring database pairs (D, D'), i.e., $|D \backslash D' \cup D' \backslash D| = 1$,

$$\Pr[M(D) \in O] \leq e^{\epsilon} \Pr[M(D') \in O]. \tag{1}$$

The privacy parameter ϵ is also known as privacy budget. A classic mechanism to achieve DP is the Laplace mechanism. We present the matrix form of Laplace mechanism here.

Theorem 2.1 (Laplace mechanism [4, 16]). *Given an $l \times n$ workload matrix $\mathbf{W}$ and a data vector $\mathbf{x}$, the Laplace Mechanism $\mathcal{L}_b$ outputs $\mathcal{L}_b(\mathbf{W}, \mathbf{x}) = \mathbf{W}\mathbf{x} + Lap(b)^l$ where $Lap(b)^l$ is a vector of l i.i.d. samples from a Laplace distribution with scale b. If $b \geq \frac{\|\mathbf{W}\|_1}{\epsilon}$, where $\|\mathbf{W}\|_1$ denotes the L_1 norm of $\mathbf{W}$, then $\mathcal{L}_b(\mathbf{W}, \mathbf{x})$ satisfies ϵ-DP.*

Li et al. [16] present the matrix mechanism, which first applies a DP mechanism, M, on a new strategy matrix $\mathbf{A}$, and then post-processes the noisy answers to the queries in $\mathbf{A}$ to estimate the queries in $\mathbf{W}$. This mechanism aims to achieve a smaller error than directly applying the mechanism M on $\mathbf{W}$. We will use the Laplace mechanism $\mathcal{L}_b$ to illustrate matrix mechanism.

Definition 2.2 (Matrix Mechanism (MM) [16]). Given an $l \times n$ workload matrix $\mathbf{W}$, a $p \times n$ strategy matrix $\mathbf{A}$, and the Laplace mechanism $\mathcal{L}_b(\mathbf{A}, \mathbf{x})$ that answers $\mathbf{A}$ on $\mathbf{x}$, the matrix mechanism $\mathcal{M}_{\mathbf{A}, \mathcal{L}_b}$ outputs the following answer: $\mathcal{M}_{\mathbf{A}, \mathcal{L}_b}(\mathbf{W}, \mathbf{x}) = \mathbf{W}\mathbf{A}^{+}\mathcal{L}_b(\mathbf{A}, \mathbf{x})$, where $\mathbf{A}^{+}$ is the Moore-Penrose pseudoinverse of $\mathbf{A}$.

Intuitively, each workload query in $\mathbf{W}$ can be represented as a linear combination of strategy queries in $\mathbf{A}$, i.e., $\mathbf{W}\mathbf{x} = \mathbf{W}\mathbf{A}^{+}(\mathbf{A}\mathbf{x})$. We denote $\mathcal{L}_b(\mathbf{A}, \mathbf{x})$ by $\tilde{\mathbf{y}}$ and $\mathcal{M}_{\mathbf{A}, \mathcal{L}_b}$ by $\tilde{\mathbf{z}}$. As the MM post-processes the output of a DP mechanism [5], it also satisfies ϵ-DP.

Proposition 2.1 ([16]). If $b \geq \frac{\|\mathbf{A}\|_1}{\epsilon}$, then $\mathcal{M}_{\mathbf{A}, \mathcal{L}_b}$ satisfies ϵ-DP.

Instead of choosing an appropriate ϵ, data analysts may simply specify accuracy requirements for their queries. We consider two popular error specifications for DP mechanisms.

Definition 2.3. Given a $l \times n$ workload matrix $\mathbf{W}$ and a DP mechanism M, (i) the α^2-expected total squared error bound [16] is

$$\mathbb{E}[\|\mathbf{W}\mathbf{x} - M(\mathbf{W}, \mathbf{x})\|_2^2] \leq \alpha^2 \tag{2}$$

and (ii) the (α, β)-worst error bound [9] is defined as

$$\Pr[\|\mathbf{W}\mathbf{x} - M(\mathbf{W}, \mathbf{x})\|_\infty \geq \alpha] \leq \beta. \tag{3}$$

The error for the matrix mechanism is $\|\mathbf{W}\mathbf{A}^{+}Lap(b)^{l}\|$, which is independent of the data. This allows a direct estimation of the error bound without running the algorithm on the data. For example, Ge et al. [9] provide a loose bound for the noise parameter in the matrix mechanism to achieve an (α, β)-worst error bound.

Theorem 2.2 ([9])**.** *The matrix mechanism $\mathcal{M}_{\mathrm{A},\mathcal{L}_b}$ satisfies the (α, β)-worst error bound, if*

$$b \le b_L = \frac{\alpha\sqrt{\beta/2}}{\|\mathbf{W}\mathbf{A}^{+}\|_F} \tag{4}$$

where $\|\cdot\|_F$ is the Frobenius norm.

When we set b to this loose bound b_L, the privacy budget consumed by this mechanism is $\frac{\|\mathrm{A}\|_1}{b_L}$. To minimize the privacy cost, Ge et al. [9] conduct a continuous binary search over noise parameters larger than b_L. The filtering condition for this search is the output of a Monte Carlo (MC) simulation for the error term $\|\mathbf{W}\mathbf{A}^{+}Lap(b)^{l}\|_\infty$ (i.e., if the sampled error exceeds α with a probability $\le \beta$).

3 SYSTEM DESIGN

We design an interactive inference engine with a built-in cache, *CacheDP*, that supports data analysts in answering data exploration queries with sufficient accuracy, without requiring them to have any differential privacy knowledge. The data owner instantiates an unmodified relational DBMS such as MySQL, with a database that includes sensitive data. To complete the setup stage, the data owner also provides a total privacy budget $\mathcal{B}$ to our system. At runtime, the data analyst inputs a workload query $\mathbb{W}$, and an (α, β) accuracy requirement that the query should satisfy, to *CacheDP*. Our system interacts with the DBMS, via an SQL interface, and a cache C, to return a differentially private workload response $\tilde{\mathbf{z}}$, which satisfies this accuracy requirement, to the analyst. Each workload response consumes a privacy budget ϵ, out of $\mathcal{B}$, and the goal of *CacheDP* is to reduce ϵ by using our cache, which stores historical noisy responses. We provide an overview of our system design in this section, while motivating our description through design challenges. Our system follows a modular design, in order to enable DP experts to develop new cache-aware, problem-specific modules in the future.

3.1 Cache Structure Overview

Our cache stores previously released noisy DP responses and related parameters; it does not store any sensitive ground truth data. Moreover, the cache does not interact directly with the DBMS at all. Therefore, the cache design evolves independently of the DBMS or other alternative data storage systems. We consider two design questions: (i) which queries and their noisy responses should be stored in the cache; and (ii) what other parameters are needed?

A naive cache design simply stores all historical workloads, their accuracy requirements and noisy responses $[(\mathbb{W}_1, \alpha_1, \beta_1, \tilde{\mathbf{z}}_1), \ldots, (\mathbb{W}_t, \alpha_t, \beta_t, \tilde{\mathbf{z}}_t)]$. When a new workload $(\mathbb{W}_{t+1}, \alpha_{t+1}, \beta_{t+1})$ comes in, the system first infers a response $\tilde{\mathbf{z}}'_{t+1}$ from the cache and its error bound α'_{t+1}. If its error bound is worse than the accuracy requirement, i.e., $\alpha'_{t+1} \ge \alpha_{t+1}$, then additional privacy budget ϵ_{t+1} needs to be spent to improve $\tilde{\mathbf{z}}'_{t+1}$ to $\tilde{\mathbf{z}}_{t+1}$. This additional privacy cost ϵ_{t+1} should be smaller than a DP mechanism that does not use historical query answers.

This cache design is used in Pioneer [24], but it has several drawbacks. First, this design results in a cache size that linearly increases with the number of workload queries. Second, we will not be able to compose and reuse cached past responses to overlapping workloads ($\mathbb{W}_{t-k} \cap \mathbb{W}_t \neq \emptyset$). Simply put, this design works with only simple DP mechanisms, which answer the data analyst-supplied workloads directly with noisy responses. For instance, Pioneer [24] considers only single query workloads and the Laplace mechanism. We seek to design a reusable cache that can work with complex DP mechanisms, and in particular, the matrix mechanism. Thus, we need to structure our cache such that cached queries and their noisy responses can be reused efficiently, in terms of the additional privacy cost and run time, while limiting the cache size.

Our key insight is that the strategy matrices in Matrix Mechanism in Def 2.2 can be chosen from a structured set. So, we store noisy responses to the matrix that the mechanism answers directly (the strategy matrix), instead of storing noisy responses that are post-processed and returned to the data analyst (the workload matrix). If all the strategy matrices share a similar structure, in other words, many similar queries, then we need to only track a limited set of queries in our cache. Relatedly, since the (α, β) accuracy requirements for different workload matrices can only be composed through a loose union bound, we instead track the noise parameters that are used to answer the associated strategy matrices. Thus, in our cache, we store the strategy queries, the noisy strategy query responses and the noise parameters.

This cache design motivates us to consider a global strategy matrix $\mathbb{A}^*$ for the cache that can support all possible workloads. Importantly, for a given workload matrix $\mathbb{W}$, we present a strategy transformer (ST) module to generate an instant strategy matrix, denoted by $\mathbb{A}$, such that each instant strategy matrix is contained in the global strategy matrix, i.e., $\mathbb{A} \subseteq \mathbb{A}^*$. In this design, the cache tracks each strategy entry $\mathbb{a} \in \mathbb{A}^*$, with its noisy response, its noise parameter, and the timestamp.

Definition 3.1 (Cache Structure)**.** Given a global strategy matrix $\mathbb{A}^*$ over the full domain $dom(\mathcal{R})$, a cache for differentially private counting queries is defined as

$$C_{\mathbb{A}^*} = \{\ldots, (\mathbb{a}, b, \tilde{y}, t), \ldots | \mathbb{a} \in \mathbb{A}^*\}, \tag{5}$$

where b and $\tilde{y}$ are the latest noise parameter and noisy response for the strategy query $\mathbb{a}$, and t is the time stamp for the latest update of $\mathbb{a}$. At beginning, all entries are initialized as $(\mathbb{a}, -, -, 0)$, where '$-$' denotes invalid values. We use C to represent the set of entries with valid noisy responses and $t > 0$.

In this work, we consider a hierarchical structure, or k-ary tree, for $\mathbb{A}^*$, which is a popular and effective strategy matrix for MM [16] with an expected worst error of $O(\log^3 n)$, where n is the domain size. Figure 1 shows the global strategy matrix as a binary tree decomposition of a small integer domain $[0, 8)$.

3.2 Strategy Transformer (ST) Overview

We outline the Strategy Transformer (ST) module, which is commonly used by all of our cache-aware DP modules. The ST module consists of two components: a Strategy Generator (SG) and a Full-rank Transformer (FRT). Prior work [16] uses the global strategy $\mathbb{A}^*$, which has a high $\|\mathbb{A}^*\|_1$. Given an input $\mathbb{W}$, the SG selects a basic

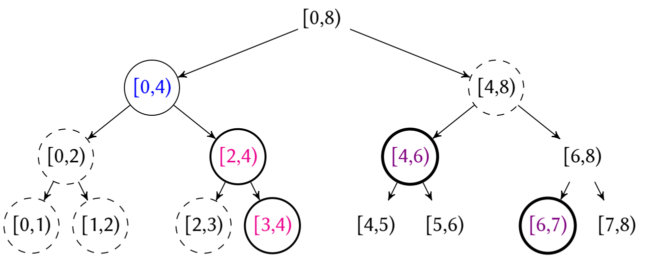


Figure 1: A global strategy $\mathbb{A}^*$ in a binary tree decomposition for an integer domain $[0, 8)$. Workloads include $\mathbb{W}_1 = \{[0, 7)\}$, $\mathbb{W}_2 = \{[2, 6), [3, 7)\}$. Strategy nodes unique to $\mathbb{A}_1$ are in blue text, those unique to $\mathbb{A}_2$ are in magenta text, whereas those in both $\mathbb{A}_1$ and $\mathbb{A}_2$ are in purple text. The dashed nodes are output by the PQ module (Section 5.3), for $\mathbb{A}_2$.

instant strategy $\mathbb{A} \subseteq \mathbb{A}^*$, with a low $\|\mathbb{A}^*\|_1$, among other criteria. The cache is not searched while generating $\mathbb{A}$. We generate instant strategies for example workloads below.

Example 3.1. In Figure 1, for an integer domain $[0, 8)$, we show a binary tree decomposition for its global strategy $\mathbb{A}^*$. This strategy consists of $(2^3 + 2^2 + 2^1 + 1)$ row counting queries (RCQs), where each RCQ corresponds to the counting query with the predicate range indicated by a node in the tree. We use $\mathbb{A}^*_{[a,b)}$ to denote the RCQ with a range $[a, b)$ in the global strategy matrix.

The first workload $\mathbb{W}_1$ consists of a single query with a range predicate $[0, 7)$. Its answer can be composed by summing over noisy responses to three RCQs in the global strategy matrix, $(\mathbb{A}^*_{[0,4)}, \mathbb{A}^*_{[4,6)}, \mathbb{A}^*_{[6,7)})$. The second workload $\mathbb{W}_2$ has two queries with range predicates $([2, 6), [3, 7))$. It can be answered using $\mathbb{A}_2 = (\mathbb{A}^*_{[2,4)}, \mathbb{A}^*_{[4,6)}, \mathbb{A}^*_{[3,4)}, \mathbb{A}^*_{[6,7)})$. We detail the strategy generation in Example 5.1.

We observe that the RCQs $\mathbb{A}^*_{[4,6)}$ and $\mathbb{A}^*_{[6,7)}$ are common to both $\mathbb{A}_1$ and $\mathbb{A}_2$, thus our cache-aware DP mechanisms can potentially reuse their noisy responses to answer $\mathbb{A}_2$. □

The accuracy analysis of the matrix mechanism only holds over full rank strategy matrices, however, the instant strategy $\mathbb{A}$ may be a very sparse matrix over the full domain, and thus, may not be full rank. We address this challenge in the FRT module, by mapping the instant strategy $\mathbb{A}$, workload $\mathbb{W}$, data vector $\mathbb{x}$, to a compact, full-rank, efficient representation, resulting in $\mathbf{A}$, $\mathbf{W}$ and $\mathbf{x}$ respectively. Thus for an input $\mathbb{W}, \mathbb{x}$, the ST module outputs $(\mathbf{A}, \mathbf{W}, \mathbf{x})$. Since the cache entries should be uniquely addressable, the raw data vector $\mathbb{x}$ and strategy $\mathbb{A}$ are used to index the cache.

3.3 Cache-aware DP Modules

Our system supports two novel classes of cache-aware DP mechanisms: Modified Matrix Mechanism (MMM) and the Relax Privacy Mechanism (RP). Each cache-aware DP mechanism implements two interfaces (similar to APEx [9]) using the ST module outputs, namely $(\mathbf{A}, \mathbf{W}, \mathbf{x})$, as well as the cache $C_{\mathbb{A}^*}$:

- The answerWorkload interface answers a workload $\mathbf{W}$ using the cache $C_{\mathbb{A}^*}$ and an instant strategy $\mathbf{A}$ to derive fresh noisy strategy responses, using the ground truth from the DB. Each implementation of this interface also updates the cache $C_{\mathbb{A}^*}$.

Algorithm 1 *CacheDP* Overview

```
Require: Dataset $D$, Total privacy budget $\mathcal{B}$.
 1: Initialize privacy loss $B_c = 0$, cache $C_{\mathbb{A}^*} = \{(\mathbb{w}, -, -, 0) | \mathbb{w} \in \mathbb{A}^*\}$
 2: repeat
 3:     Receive $(Q, \alpha, \beta)$ from analyst
 4:     $\mathbb{W} \leftarrow$ getMatrixForm$(Q, \mathbf{x})$
 5:     $\mathbb{A}, \mathbf{A}, \mathbf{W} \leftarrow$ generateStrategy$(\mathbb{W}, \mathbb{A}^*)$
 6:     $(b, \epsilon_1) \leftarrow$ MMM.estimatePrivacyBudget$(C, \mathbf{A}, \mathbf{W}, \alpha, \beta)$
 7:     $\epsilon_2 \leftarrow$ RP.estimatePrivacyBudget$(C, \mathbf{A}, \mathbf{W}, \alpha, \beta)$
 8:     $\mathbb{A}_e, \mathbf{A}_e, \leftarrow$ SE.generateExpandedStrategy$(\mathbb{A}, C, b)$
 9:     $\epsilon_3 \leftarrow$ MMM.estimatePrivacyBudget$(C, \mathbf{A}_e, \mathbf{W}, \alpha, \beta)$
10:     Pick $(\hat{M}, \hat{\mathbf{A}})$ from (MMM/RP, $\mathbf{A}/\mathbf{A}_e$) that has smallest $\epsilon_i$
11:     if $\epsilon_i + B_c \geq \mathcal{B}$ then
12:         Answering $Q$ satisfying $(\alpha, \beta)$ will exceed $\mathcal{B}$. Reject $Q$.
13:     $z \leftarrow \hat{M}$.answerWorkload$(C, \hat{\mathbf{A}}, \mathbf{W}, \epsilon_i, \mathbf{x})$
14:     return $z$ to data analyst.
15:     $B_c \leftarrow B_c + \epsilon_i$
16: until no more $Q$ from the analysts
```

- The estimatePrivacyBudget interface estimates the minimum privacy budget ϵ required by the answerWorkload interface to achieve the (α, β) accuracy requirement.

For the first cache-aware DP mechanism, MMM, we have two additional optional modules, namely Strategy Expander (SE) and Proactive Querying (PQ), which modify the instant strategy $\mathbb{A}$ output by the basic ST module, for different purposes. The SE module expands the basic $\mathbb{A}$ with related, cached, accurate strategy rows in $C_{\mathbb{A}^*}$ to exploit constrained inference as discussed by Hay et al. [11]. The goal of this module is to further reduce the privacy cost of the basic instant strategy to answer the given workload $\mathbb{W}$. On the other hand, the PQ module is designed to fill the cache proactively, for later use by the MMM, MMM+SE, and RP mechanisms. It expands $\mathbb{A}$ with strategy queries that are absent from $C_{\mathbb{A}^*}$, without incurring any additional privacy budget over the MMM module. Therefore, it reduces the privacy cost of future workload queries.

Putting it all together, we state the end-to-end algorithm in Algorithm 1. First, for an input workload $(\mathbf{W}, \alpha, \beta)$, our system first uses the ST module to generate a full-rank instant strategy matrix $\mathbf{A}$ (line 5), and then executes the estimatePrivacyBudget interface, with the input tuple $(\mathbf{W}, \mathbf{A}, \alpha, \beta)$, for the MMM, MMM+SE, and RP mechanisms (line 6-9). We choose the mechanism that returns the lowest privacy cost ϵ_i (line 10). If the sum of this privacy cost with the consumed privacy budget is smaller than the total privacy budget, then the system executes the answerWorkload interface for the chosen mechanism, with the input tuple $(\mathbf{W}, \hat{\mathbf{A}}, \epsilon_i)$ (line 13). The consumed privacy budget will increase by ϵ_i (line 15). (The PQ module does not impact the cost estimation for MMM, it only extends the strategy matrix $\mathbf{A}$ to be answered.) We present the MMM in Section 4, the common ST module and the MMM optional modules (SE, PQ) in Section 5, and the RP mechanism in Section 6.

Theorem 3.1. *CacheDP, as defined in Algorithm 1, satisfies $\mathcal{B}$-DP.*

4 MODIFIED MATRIX MECHANISM (MMM)

We describe our core cache-aware DP mechanism, namely the Modified Matrix Mechanism. We wish to answer a workload $\mathbf{W}$ with an

(α, β)-accuracy requirement using a cache $C_{\mathbf{A}^*}$ and an instant strategy $\mathbf{A} \subseteq \mathbf{A}^*$, while minimizing the privacy budget ϵ. First, we intuit the mechanism design. Second, we describe the answerWorkload interface, which answers $\mathbf{W}$ through $\mathbf{A}$, under optimal parameters. Third, we describe the EstimatePrivacyBudget interface, which derives an optimal ϵ and other parameters for the former interface.

4.1 MMM Overview

The cacheless matrix mechanism (Definition 2.2) perturbs the ground truth response to the strategy, that is $\mathbf{Ax}$, with the noise vector freshly drawn from $Lap(b)^{|\mathbf{A}|}$ to obtain $\tilde{\mathbf{y}} = \mathbf{Ax} + Lap(b)^{|\mathbf{A}|}$. An input workload is then answered using $\mathbf{WA}^+\tilde{\mathbf{y}}$. As we discussed in the background, in an accuracy-aware DP system such as APEx [9], the noise parameter b is calibrated, first through a loose bound b_L and then to a tighter noise parameter b_T, such that the workload response above meets the (α, β)-accuracy requirement. This spends a privacy budget $\frac{\|\mathbf{A}\|_1}{b_T}$ (Proposition 2.1).

In MMM, we seek to reduce the privacy budget spent by using the cache C. Given an instant strategy matrix $\mathbf{A} \subseteq \mathbf{A}^*$, we first lookup the cache for any rows in the strategy matrix $\mathbf{A}$. Note that not all rows in $\mathbf{A}$ have their noisy responses in the cache. The cache may contain noisy responses for some rows of $\mathbf{A}$, given by $C \cap \mathbf{A}$, whereas other rows in $\mathbf{A}$ may not have cached responses. A preliminary approach would be to simply reuse all cached strategy responses, and obtain noisy responses for non-cached strategy rows by expending some privacy budget through naive MM. However, some cached responses may be too noisy and thus including them will lead to a higher privacy cost than the cacheless MM.

Our key insight is that by reusing noisy responses for accurately cached strategy rows, MMM can ultimately use a smaller privacy budget for all other strategy rows as compared to MM without cache while satisfying the accuracy requirements. Thus, out of all cached strategy rows $C \cap \mathbf{A}$, MMM identifies a subset of accurately cached strategy rows $\mathbf{F} \subseteq C \cap \mathbf{A}$ that can be directly answered using their cached noisy responses, without spending any privacy budget. MMM only spends privacy budget on the remaining strategy rows, namely on $\mathbf{P} = \mathbf{A} - \mathbf{F}$. We refer to $\mathbf{F}$ and $\mathbf{P}$ as the free strategy matrix and the paid strategy matrix respectively. MMM consists of two interfaces as indicated by Algorithm 2: (i) answerWorkload and (ii) estimatePrivacyBudget. The second interface seeks the best pair of free and paid strategy matrices $(\mathbf{F}, \mathbf{P})$ that use the smallest privacy budget ϵ to achieve (α, β)-accuracy requirement. The first interface will make use of this parameter configuration $(\mathbf{F}, \mathbf{P}, \epsilon)$ to generate noisy responses to the workload.

4.2 Answer Workload Interface

We present the first interface answerWorkload for the MMM. We recall that this interface is always called after the estimatePrivacyBudget interface which computes the best combination of free and paid strategy matrices and their corresponding privacy budget $(\mathbf{F}, \mathbf{P}, b_\mathbf{P}, \epsilon)$. As shown in Algorithm 2, the answerWorkload interface first calls the proactive module (Section 5.3). If this module is turned on, $\mathbf{P}$ will be expanded for the remaining operations. Then this interface will answer the paid strategy matrix $\mathbf{P}$ using Laplace mechanism with the noise parameter $b_\mathbf{P}$. We have $b_\mathbf{P} = \frac{\|\mathbf{P}\|_1}{\epsilon}$, to ensure ϵ-DP (Line 4). Then, it updates the corresponding entries in the cache $C_{\mathbb{A}^*}$ (Line 5). In particular, for each query $\mathbf{w} \in \mathbf{P}$, we update its corresponding noisy parameter, noisy response, and timestamp in $C_{\mathbb{A}^*}$ to $b_\mathbf{P}$, $\tilde{y}$, and the current time. After obtaining the fresh noisy responses $\tilde{\mathbf{y}}_\mathbf{P}$ for the paid strategy matrix, this interface pulls the cached responses $\tilde{\mathbf{y}}_\mathbf{F}$ for the free strategy matrix from the cache and concatenate them into $\tilde{\mathbf{y}}$ according to their order in the instant strategy $\mathbf{A}$ (Lines 6-7). Finally, this interface returns a noisy response to the workload $\mathbf{WA}^+\tilde{\mathbf{y}}$, and its privacy cost ϵ.

Algorithm 2 MMM main interfaces and supporting functions

```
 1: function answerWorkload( $C, \mathbf{A}, \mathbf{W}, \epsilon, \mathbf{x}$)
 2:     $(\mathbf{F}, \mathbf{P}, b_\mathbf{P}, \epsilon)$ from pre-run EstimatePrivacyBudget($C, \mathbf{A}, \mathbf{W}, \alpha, \beta$)
 3:     (Optional) Expand $\mathbf{P}$ with PQ module (Section 5.3)
 4:     $\tilde{\mathbf{y}}_\mathbf{P} \leftarrow \mathbf{Px} + Lap(b_\mathbf{P})^{|\mathbf{P}|}$   ▷ we have $b_\mathbf{P} = \frac{\|\mathbf{P}\|_1}{\epsilon}$
 5:     Update cache $C_{\mathbb{A}^*}$ with $(\mathbf{P}, b_\mathbf{P}, \tilde{\mathbf{y}}_\mathbf{P}, t = \text{current time})$
 6:     $\tilde{\mathbf{y}}_\mathbf{F} \leftarrow [(\mathbf{w}, b, \tilde{y}, t) \in C | \mathbf{w} \in \mathbf{F}]$   ▷ free cached responses for $\mathbf{F}$
 7:     $\tilde{\mathbf{y}} \leftarrow \tilde{\mathbf{y}}_\mathbf{F} || \tilde{\mathbf{y}}_\mathbf{P}$   ▷ concatenate noisy responses for $\mathbf{A}$.
 8:     return $\mathbf{WA}^+\tilde{\mathbf{y}}, \epsilon$

 9: function EstimatePrivacyBudget($C, \mathbf{A}, \mathbf{W}, \alpha, \beta$)
10:     Set upper bound $b_\top = \frac{\|\mathbf{A}\|_1}{\epsilon_\perp}$   ▷ $\epsilon_\perp$ is the budget precision
11:     Set loose bound $b_L = \frac{\alpha\sqrt{\beta/2}}{\|\mathbf{WA}^+\|_F}$   ▷ Theorem 2.2 (without cache)
12:     $\mathbf{b} \leftarrow [(\mathbf{w}, b, \tilde{y}, t) \in C \mid \mathbf{w} \in \mathbf{A} \cap C, b > b_L] \cup [b_L]$
13:     $b_D \leftarrow$ binarySearch(sort($\mathbf{b}$), checkAccuracy($\cdot, C, \mathbf{A}, \mathbf{W}, \alpha, \beta$))   ▷
    Search $b_D$ in the discrete space
14:     $\mathbf{F} \leftarrow [c.\mathbf{a} \in C \mid c.\mathbf{a} \in \mathbf{A} \cap C, c.b < b_\mathbf{P}]$ and $\mathbf{P} \leftarrow \mathbf{A} - \mathbf{F}$
15:     $b_\mathbf{P} \leftarrow$ binarySearch($[b_D, b_\top]$, checkAccuracy($\cdot, C, \mathbf{A}, \mathbf{W}, \alpha, \beta$))
    ▷ Search $b_\mathbf{P}$ in a continuous space
16:     return ( $\mathbf{F}, \mathbf{P}, b_\mathbf{P}, \frac{\|P\|_1}{b_\mathbf{P}}$)
```

Proposition 4.1. The AnswerWorkload interface of MMM (Algorithm 2) satisfies ϵ-DP, where ϵ is the output of this interface.

As the final noisy response vector $\tilde{\mathbf{y}}$ to the strategy $\mathbf{A}$ is concatenated from $\tilde{\mathbf{y}}_\mathbf{F}$ and $\tilde{y}_\mathbf{P}$, its distribution is equivalent to a response vector perturbed by a vector of Laplace noise with parameters: $\mathbf{b} = \mathbf{b}_\mathbf{F} || \mathbf{b}_\mathbf{P}$, where $\mathbf{b}_\mathbf{F}$ is a vector of noise parameters for the cached entries in $\mathbf{F}$ with length $|\mathbf{F}|$ and $\mathbf{b}_\mathbf{P}$ is a vector of the same value $b_\mathbf{P}$ with length $|\mathbf{P}|$. This differs from the standard matrix mechanism with a single scalar noise parameter. We derive its error term next.

Proposition 4.2. Given an instant strategy $\mathbf{A} = (\mathbf{F}||\mathbf{P})$ with a vector of k noise parameters $\mathbf{b} = \mathbf{b}_\mathbf{F}||\mathbf{b}_\mathbf{P}$, the error to a workload $\mathbf{W}$ using the AnswerWorkload interface of MMM (Algorithm 2) is

$$\|\mathbf{WA}^+ Lap(\mathbf{b})\| \tag{6}$$

where $Lap(\mathbf{b})$ draws independent noise from $Lap(\mathbf{b}[1]), \ldots, Lap(\mathbf{b}[k])$ respectively. We can simplify its expected total square error as

$$\|\mathbf{WA}^+ diag(\mathbf{b})\|_F^2 \tag{7}$$

where $diag(\mathbf{b})$ is a diagonal matrix with $diag(\mathbf{b})[i, i] = \mathbf{b}[i]$.

4.3 Estimate Privacy Budget Interface

The second interface EstimatePrivacyBudget chooses the free and paid strategy matrices and the privacy budget to run the first interface for MMM. This corresponds to the following questions:

(1) Which cached strategy rows out of $C \cap \mathbf{A}$ should be included in the free strategy matrix $\mathbf{F}$? The choice of $\mathbf{F}$ directly determines the paid strategy matrix $\mathbf{P}$ as $\mathbf{A} - \mathbf{F}$.

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \mathbf{A} = \left[\begin{array}{ccc} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right], \mathbf{b} = \left[\begin{array}{c} b_c \\ \hline b \\ b \end{array}\right], \mathbf{x}_1 = \left[\begin{array}{c} \mathbb{x}[0,4) \\ \hline \mathbb{x}[4,6) \\ \mathbb{x}[6,7) \end{array}\right]$$

Figure 2: Consider $\mathbb{W}_1 = \{[0,7)\}$ with its corresponding mapped workload matrix, instant strategy, noise vector, and data vector. Reusing a cached response for the first row with noise parameter b_c requires a smaller noise parameter b (and hence a bigger privacy budget) for the other rows than the cacheless MM to achieve the same accuracy level.

(2) Given $\mathbf{P}$ and $b_{\mathbf{P}}$, the privacy budget paid by MMM is given by $\epsilon = \|\mathbf{P}\|_1/b_{\mathbf{P}} = \|\mathbf{A}-\mathbf{F}\|_1/b_{\mathbf{P}}$. To minimize this privacy budget, what is the maximum noise parameter value $b_{\mathbf{P}}$ that can be used to answer $\mathbf{P}$ while meeting the accuracy requirement?

A baseline approach to the first question is to simply set $\mathbf{F} = C \cap \mathbf{A}$, that is, we reuse all cached strategy responses. This approach may reuse inaccurate cached responses with large noise parameters, which results in a larger ϵ (or a smaller $b_{\mathbf{P}}$) to achieve the given accuracy requirement than answering the entire $\mathbf{A}$ by resampling new noisy responses without using the cache.

Example 4.1. Continuing with Example 3.1, we have an instant strategy $\mathbf{A}$ for the workload $\mathbb{W}_1$ with range predicate $[0,7)$ mapped to a partitioned domain $\{[0,4),[4,6),[6,7)\}$. The mapped workload and instant strategy are shown in Figure 2. For simplicity, we use the expected square error to illustrate the drawback of the baseline approach, but the same reasoning applies to (α,β)-worst error bound. Without using the cache, when we set $\mathbf{b} = [10,10,10]$, we achieve an expected error $\|\mathbf{WA}^{+}diag(\mathbf{b})\|_F^2 = 300$ for the workload $\mathbf{W}$. Suppose the cache has an entry for the first RCQ $[0,4)$ of the strategy and a noise parameter $b_c = 15$. Using this cached entry, the noise vector becomes $\mathbf{b} = [15, b_{\mathbf{P}}, b_{\mathbf{P}}]$, and the expected square error is $\|\mathbf{WA}^{+}diag(\mathbf{b})\|_F^2 = 15^2 + 2b_{\mathbf{P}}^2$. To achieve the same or a smaller error than the cacheless MM, we need to set $b_{\mathbf{P}} \leq \sqrt{(300-15^2)/2} \approx 6.12$ for the remaining entries in the strategy. This tighter noise parameter $b_{\mathbf{P}}$ corresponds to a larger privacy budget. □

4.3.1 Privacy Cost Optimizer. We formalize the two aforementioned questions as an optimization problem, subject to the accuracy requirements, as follows.

Cost estimation (CE) problem: Given a cache C and an instant strategy matrix $\mathbf{A}$, determine $\mathbf{F} \subseteq (\mathbf{A} \cap C)$ (and $\mathbf{P} = \mathbf{A} - \mathbf{F}$) and $b_{\mathbf{P}} \in [b_L, b_{\top}]$ that minimizes the paid privacy budget $\epsilon = \frac{\|\mathbf{P}\|_1}{b_{\mathbf{P}}}$ subject to accuracy requirement:

$$\|\mathbf{WA}^{+}diag(\mathbf{b_F}||\mathbf{b_P})\|_F^2 \leq \alpha^2 \text{ or}$$
$$\Pr[\|\mathbf{WA}^{+}Lap(\mathbf{b_F}||\mathbf{b_P})\|_\infty \geq \alpha] \leq \beta.$$

In this optimization problem, the lower bound for $b_{\mathbf{P}}$ is the loose bound for the cacheless MM (Equation (4)), and the upper bound $b_{\top}$ is $\frac{\|A\|_1}{\epsilon_{\perp}}$, where $\epsilon_{\perp}$ is the smallest possible privacy budget.

In a brute-force solution to this problem, we can search over all possible pairs of $\mathbf{F} \subseteq (\mathbf{A} \cap C)$ and $b_{\mathbf{P}} \in [b_L, b_{\top}]$, and check whether every possible pair of $(\mathbf{F}, b_{\mathbf{P}})$ can lead to an accurate response. In this solution, the search space for $\mathbf{F}$ will be $O(2^{|\mathbf{A} \cap C|})$ and thus the total search space will be $O\left(2^{|\mathbf{A} \cap C|} \cdot \log_2(|[b_L, b_{\top}]|)\right)$ if we apply binary search within $[b_L, b_{\top}]$. Hence, we need another way to efficiently determine optimal values for $(\mathbf{F}, b_{\mathbf{P}})$.

4.3.2 Simplified Privacy Cost Optimizer. We present a simplification to arrive at a much smaller search space for $(\mathbf{F}, b_{\mathbf{P}})$, while ensuring that $b_{\mathbf{P}}$ improves over the noise parameter of the cacheless MM. We observe that, if we perturb the paid strategy matrix with noise parameter $b_{\mathbf{P}}$ and choose cached entries with noise parameters smaller than $b_{\mathbf{P}}$, we will have a smaller error than a cacheless MM with a noise parameter $b = b_{\mathbf{P}}$ for all the queries in the strategy matrix. This motivates us to consider the following search space for $\mathbf{F}$. When given $b_{\mathbf{P}}$, we choose a free strategy matrix fully determined by this noise parameter:

$$\mathbf{F}_{b_{\mathbf{P}}} = \{c.\mathbf{a} \in C \mid c.\mathbf{a} \in C \cap \mathbf{A}, c.b \leq b_{\mathbf{P}}\}, \tag{8}$$

and formalize a simplified optimization problem.

Simplified CE problem: Given a cache C and an instant strategy matrix $\mathbf{A}$, determine $b_{\mathbf{P}} \in [b_L, b_{\perp}]$ (and $\mathbf{F} = \mathbf{F}_{b_{\mathbf{P}}}$, $\mathbf{P} = \mathbf{A} - \mathbf{F}$) that minimizes the paid privacy budget $\epsilon = \frac{\|\mathbf{P}\|_1}{b_{\mathbf{P}}}$ subject to:

$$\|\mathbf{WA}^{+}diag(\mathbf{b_F}||\mathbf{b_P})\|_F^2 \leq \alpha^2 \text{ or}$$
$$\Pr[\|\mathbf{WA}^{+}Lap(\mathbf{b_F}||\mathbf{b_P})\|_\infty \geq \alpha] \leq \beta.$$

Theorem 4.1. *The optimal solution to simplified CE problem incurs a smaller privacy cost ϵ than the privacy cost $\epsilon_{\mathbf{F}=\emptyset}$ of the matrix mechanism without cache, i.e., MMM with $\mathbf{F} = \emptyset$.*

4.3.3 Algorithm for Simplified CE Problem. We present our search algorithm to find the best solution to the simplified CE problem, shown in the ESTIMATEPRIVACYBUDGET function of Algorithm 2. In our extended paper, we visualize our searches through the cached noise parameters. First, we setup the upper and lower bounds for the noise parameter $b_{\mathbf{P}}$ for the simplified CE problem (Lines 10-11).

Step 1: Discrete search for $b_{\mathbf{P}}$. We first search $b_{\mathbf{P}}$ from the existing noise parameters in the cached strategy rows $\mathbf{A} \cap C$ that are greater than b_L (Line 12). We also include b_L in this noise parameter list $\mathbf{b}$. Next, we sort the noise parameter list $\mathbf{b}$ and conduct a binary search in this sorted list to find the largest possible $b_D \in \mathbf{b}$ that meets the accuracy requirement (Line 13). During this binary search, to check if a given $b_{\mathbf{P}}$ achieves (α,β)-accuracy requirement, we run the function CHECKACCURACY. This function first places all the cached entries with noise parameter smaller than $b_{\mathbf{P}}$ into $\mathbf{F}$ and the remaining entries of the strategy into $\mathbf{P}$. Then it runs an MC simulation of the error $\mathbf{WA}^{+}Lap(\mathbf{b_F}||\mathbf{b_P})$ (Proposition 4.2). If a small number of the simulated error vectors have a norm bigger than α, then this paid noise vector $b_{\mathbf{P}}$ achieves (α,β)-accuracy guarantee. This MC simulation differs from a traditional one [9] which makes no use of the cache and has only a single scalar noise value for all entries of the strategy. On the other hand, if the accuracy requirement is α^2-expected total square error, we simply check if $\|\mathbf{WA}^{+}diag(\mathbf{b_F}||\mathbf{b_P})\|_2^2 \leq \alpha^2$.

Step 2: Refining $b_{\mathbf{P}}$ in a continuous space. We observe that we may further increase $b_{\mathbf{P}}$, by examining the interval between b_D, which is the output from the discrete search, and the next largest cached noise parameter, denoted by $\top_C = b_{D+1}$. If $\top_C$ does not exist, then we set $\top_C = b_{\top}$. We conduct a binary search in a continuous domain $[b_D, \top_C]$ (Line 15). This continuous search does not impact the free strategy matrix $\mathbf{F}$ obtained from the discrete search, as the chosen noise parameter will be strictly smaller than b_{D+1}. This search outputs a noise parameter $b_{\mathbf{P}}$. Finally, this function returns $b_{\mathbf{P}}$, the privacy budget $\epsilon = \frac{\|\mathbf{P}\|_1}{b_{\mathbf{P}}}$, as well as the free and paid strategy matrices outputted from the discrete search.

The search space for this simplified CE problem is $O(\log_2(|[b_L, b_\top]|))$. We only need to sort the cached matrix once, which costs $O(n_c \cdot \log(n_c))$, where $n_c = |\mathbf{A} \cap C|$. Hence, this approach significantly improves the brute-force search solution for the CE problem.

5 STRATEGY MODULES

In this section, we first present the strategy transformer (ST), which is used by all of our cache-aware DP mechanisms. We then present two optional modules for MMM: the Strategy Expander (SE) and Proactive Querying (PQ). Due to space constraints, all detailed algorithms for this section are included in the full paper [18].

5.1 Strategy Transformer

The ST module selects an instant strategy from the given global strategy $\mathbb{A} \subseteq \mathbb{A}^*$ based on the workload $\mathbb{W}$. Since our cache-aware MMM and RP modules build on the matrix mechanism, we require a few basic properties for this instant strategy $\mathbb{A}$ to run the former mechanisms, with good utility. First, the strategy $\mathbb{A}$ should be a support to the workload $\mathbb{W}$ [16], that is, it must be possible to represent each query in $\mathbb{W}$ as a linear combination of strategy queries in $\mathbb{A}$. In other words, there exists a solution matrix $\mathbb{X}$ to the linear system $\mathbb{W} = \mathbb{X}\mathbb{A}$. Second, $\mathbb{A}$ should have a low l_1 norm, such that the privacy cost $\epsilon = \frac{\|\mathbb{A}\|_1}{b}$ for running MM is small, for a given a noise parameter b (Proposition 2.1). Third, using noisy responses to $\mathbb{A}$ to answer $\mathbb{W}$ should incur minimal noise compounding [11]. We thus present the strategy generator (SG) component, to address all of these requirements. The strategy generator only uses the global strategy $\mathbb{A}^*$, and does not use the cached responses, to generate an instant strategy $\mathbb{A}$ for the workload $\mathbb{W}$.

Last, we require that $\mathbb{A}$ must be mapped to a full rank matrix $\mathbf{A}$, such that $\mathbf{A}^+\tilde{\mathbf{y}}$ is the estimate of the mapped data vector $\mathbf{x}$ that minimizes the total squared error given the noisy observations $\tilde{\mathbf{y}}$ of the strategy queries $\mathbf{A}$ [16, Section 4]. We present a full-rank transform (FRT) component to address this last requirement. The ST module runs the SG and FRT components sequentially.

5.1.1 Strategy Generator. Consider using the global strategy $\mathbb{A}^*$ as follows: to answer the first workload, we obtain the noisy strategy responses for all nodes on the tree, thereby fully populating the cache. Cached noisy responses can be reused for future workloads. Though $\mathbb{A}^*$ supports all possible counting queries over $dom(\mathcal{R})$, it has a very high norm $\|\mathbb{A}^*\|$, equal to the tree height $\log_k(n) + 1$, where n is the full domain size. Thus, answering the first workload would require spending a high upfront privacy budget, which may not be amortized across future workloads, as they may focus on a small part of the domain with higher accuracy requirements.

To obtain a low norm strategy matrix, we only choose those strategy queries from $\mathbb{A}^*$ that support the workload $\mathbb{W}$. Intuitively, we wish to fill the cache with noisy responses to as many strategy queries as possible, thus we should bias our strategy generation algorithm towards the leaf nodes of the strategy tree. However, the DP noisy responses for the strategy nodes would be added up to answer the workload, and summing up responses to a large number of strategy leaf nodes compounds the DP noise in the workload response [11]. Thus, for each query in the workload $\mathbb{W}$, we apply a top-down tree traversal to fetch the minimum number of nodes in the strategy tree (and the corresponding queries in $\mathbb{A}^*$) required to answer this workload query. Then we include all these queries into the instant strategy $\mathbb{A}$ for this workload $\mathbb{W}$. The L_1 norm of the output strategy matrix is then simply the maximum number of nodes in any path of the strategy tree, and it is upper-bounded by the tree height. We present an example strategy generation below.

Example 5.1. We continue with Example 3.1 shown in Figure 1, for an integer domain $[0, 8)$. For the single workload query $\mathbb{W}_1 = \mathbb{w} = [0, 7)$, the first iteration of our SG workload decomposition algorithm computes the overlap of $\mathbb{w}$ with its left child $c_1 = \mathbb{A}^*_{[0,4)}$ as $\mathbb{w}_{c1} = [0, 4)$ and the overlap with its right child $c_2 = \mathbb{A}^*_{[4,8)}$ as $\mathbb{w}_{c2} = [4, 7)$. The function only iterates once for the left child c_1, directly outputs that child's range $\mathbb{A}^*_{[0,4)}$, as the base condition is satisfied. In the next iteration for the right child c_2, the overlaps with both of its children are non-null ($[4, 6)$ with $\mathbb{A}^*_{[4,6)}$ and $[6, 7)$ with $\mathbb{A}^*_{[6,8)}$), and the corresponding strategy nodes are returned in subsequent iterations. Since $\mathbb{A}_1$ has no overlapping intervals, $\|\mathbb{A}_1\|_1 = 1 < \|\mathbb{A}^*\|_1$. We observe that though $\mathbb{A}^*$ is full-rank, due to the removal of strategy queries that do not support the workloads, both $\mathbb{A}_1$ and $\mathbb{A}_2$ are not full rank. □

5.1.2 Full Rank Transformer (FRT). We transform an instant strategy matrix $\mathbb{A}$ to a full rank matrix $\mathbf{A}$ by mapping the full domain $dom(\mathcal{R})$ of size n to a new partition of the full domain of $n' \leq n$ non-overlapping counting queries or buckets. The resulting partition should still support all the queries in the instant raw strategy $\mathbb{A}$ output by our SG. For efficiency, the partition should have the smallest possible number of buckets such that the transformed strategy $\mathbf{A}$ will be full rank. First, we define a domain transformation matrix $\mathbb{T}$ of size $n' \times n$ that transforms the data vector $\mathbb{x}$ over the full domain to the partitioned data vector $\mathbf{x}$, such that $\mathbf{x} = \mathbb{T}\mathbb{x}$. Using $\mathbb{T}$, we can then transform a raw $\mathbb{A}$ to a full-rank $\mathbf{A}$.

Definition 5.1 (Transformation Matrix)**.** Given a partition of n' non-overlapping buckets over the full domain $dom(\mathcal{R})$, if the ith value in $dom(\mathcal{R})$ is in the jth bucket, $\mathbb{T}[j, i] = 1$; else, $\mathbb{T}[j, i] = 0$.

Theorem 5.1. *Given a global strategy $\mathbb{A}^*$ in a k-ary tree structure, and an instant strategy $\mathbb{A} \subseteq \mathbb{A}^*$, TRANSFORMSTRATEGY outputs a strategy $\mathbf{A}$ that is full rank and supports $\mathbb{A}$.*

We present an example FRT in our full paper. The ST module finally outputs $\mathbb{A}$, $\mathbf{A}$, as well as the transformation matrix, as it can be used to transform $\mathbb{W}$. We use the full-rank versions $\mathbf{W}, \mathbf{A}$ for all invocations of the matrix mechanism (i.e. computing $\mathbf{W}\mathbf{A}^+$).

5.2 Strategy Expander

We recall that our goal with *CacheDP* is to use cached strategy responses, in order to save privacy budget on new strategy queries. Section 4 shows that MMM achieves this goal by directly reusing accurate strategy responses from the cache for the basic instant matrix, i.e., by selecting $\mathbb{F} \subseteq C \cap \mathbb{A}$. In this strategy expander (SE) module, we provide efficient heuristics to include additional cached strategy entries out of $C - \mathbb{A}$, to $\mathbb{A}$ to save more privacy budget.

Consider a strawman solution to choosing cache entries: we simply add all strategy queries from $C - \mathbb{A}$ to $\mathbb{A}$, in order to obtain an expanded strategy $\mathbb{A}_e$. The error term for the expanded strategy is given by: $\mathbf{W}\mathbf{A}_e^{+}\text{diag}(\mathbf{b}_e)$ (Proposition 4.2). In our full version of the paper [18], we discuss related work hypothesizing this strawman solution [16], and we present an example wherein the strawman solution can lead to a strategy with an increased error term. Intuitively, adding a strategy query results in changed coefficients in $\mathbf{W}\mathbf{A}_e^{+}$, that is, this added query changes the weight with which noisy responses to the original strategy queries are used to form the workload response. The added strategy query response must also be accurate, since adding a large, cached noise parameter to $\mathbf{b}_e$ will also likely increase the magnitude of the error term (recall the example in Figure 2).

A brute force approach to find the optimal $\mathbb{A}_e$ would consider all possible subsets of cache entries from $C - \mathbb{A}$ and check if the error is better than the original strategy. This induces an exponentially large search space of $O(2^{|C|})$ possible solutions for $\mathbb{A}_e$. We propose a series of efficient heuristics to obtain a greedy solution.

First, we search only the strategy queries from $C - \mathbb{A}$ that are accurate enough. Recall that the MMM.estimatePrivacyBudget interface outputs the noise parameter $b_{\mathbf{P}}$. Just as we used $b_{\mathbf{P}}$ to compute $\mathbf{F}$, we can also use it to select cache entries for $\mathbb{A}_e$ that are at least as accurate as other entries in $\mathbf{F}$. These accurate cached responses will likely improve the accuracy of the workload response. We first sort the cache entries in increasing order of the noise parameters and add each entry to $\mathbb{A}_e$ one by one until its noise parameter is greater than $b_{\mathbf{P}}$, or, we reach a maximum bound on the cached strategy size. This approach reduces the search space from $O(2^{|C|})$ to $O(|C|)$ and ensures that the additional strategy rows do not significantly increase the run-time of *CacheDP*.

Second, we ensure that each query $\mathbb{q}$ added to $\mathbb{A}_e$ is a parent or a child of an existing query $\mathbb{q}' \in \mathbb{A}$. Our heirarchical global strategy $\mathbb{A}^*$ structures cache entries, and induces relations between the cached noisy responses. The constrained inference problem focuses on minimizing the error term for multiple noisy responses, while following consistency constraints among them, as described by Hay et al. [11]. For example, if we add the strategy queries corresponding to the siblings and parent nodes of an existing query in $\mathbb{A}$, we obtain an additional consistency constraint which tends to reduce error. However, if we only added the sibling node, we would not have seen as significant (if any) improvement. This heuristic selects strategy rows that are more likely to reduce the privacy budget compared to MMM ($\epsilon_{\mathbf{P}}$).

The privacy budget for $\mathbf{A}_e$ is estimated using the MMM.estimatePrivacyBudget interface. We encapsulate SE as a module rather than integrate it with MMM, since our heuristics might fail and $\mathbf{A}_e$ might cost a higher privacy budget than the $\mathbf{A}$ used by MMM. Since Algorithm 1 chooses the answerWorkload interface for the module and strategy with the lowest privacy cost, in the above case, $\mathbf{A}_e$ is simply not used. We analyze the conditions under which our heuristics result in SE module being selected, in our full paper [18].

5.3 Proactive Querying

The proactive querying (PQ) module is an optional module for MMM. The MMM obtains fresh noisy responses only for the paid strategy matrix $\mathbf{P}$, and inserts them into the cache. The goal of the PQ module is to proactively populate the cache with noisy responses to a subset $\Delta\mathbb{P}$ out of the remaining, non-cached strategy queries of the global strategy ($\mathbb{A}^* - C - \mathbb{P}$), where $\mathbb{P}$ corresponds to the raw, non-full rank form of $\mathbf{P}$. Thus, we run the PQ module in the function MMM.answerWorkload($\cdot$) after obtaining the paid strategy matrix $\mathbf{P}$. Our cache-aware modules, including MMM, RP and SE, can use the cached noisy responses to $\Delta\mathbb{P}$ to answer future instant strategy queries. We wish to satisfy this goal without consuming any additional privacy budget over the MMM.

We first motivate key constraints for the PQ algorithm. First, we do not assume any knowledge of future workload query sequences. However, all future workload queries will be transformed into instant strategy matrices, and our cache-aware mechanisms will lookup the cache for cached strategy rows. Second, we also do not know the accuracy requirements for future workload queries. Future workloads may be asked at different accuracy requirements than the current workload. Thus, we choose to obtain responses to $\Delta\mathbb{P}$ at the highest possible accuracy requirements without spending any additional privacy budget over that required for $\mathbf{P}$ by MMM, which is $\epsilon = \frac{\|\mathbf{P}\|_1}{b_{\mathbf{P}}}$. Our key insight is to generate $\Delta\mathbb{P} \subseteq (\mathbb{A}^* - C - \mathbb{P})$ such that $\|\mathbb{P} \cup \Delta\mathbb{P}\|_1 = \|\mathbb{P}\|_1$. Therefore, answering both instant strategies ($\mathbb{P}$ and $\Delta\mathbb{P}$) with the Laplace mechanism using $b_{\mathbf{P}}$ costs no more privacy budget than simply answering $\mathbb{P}$ at $b_{\mathbf{P}}$.

Theorem 5.2. *Given a paid strategy matrix $\mathbb{P}$ our proactive strategy generation algorithm outputs $\Delta\mathbb{P}$ such that $\|\mathbb{P} \cup \Delta\mathbb{P}\|_1 = \|\mathbb{P}\|_1$.*

Our proactive generation algorithm consists of two top-down traversals of the tree. We illustrate our proactive strategy generation algorithm through the following example. Our detailed algorithm and theorem proofs are in the full paper [18].

Example 5.2. In Figure 1, we apply our proactive strategy generation function to to $\mathbb{P}_2 = \{\mathbb{A}^*_{[2,4)}, \mathbb{A}^*_{[3,4)}\}$ for $\mathbb{W}_2$ in our example sequence. Our algorithm outputs $\Delta\mathbb{P}_2 = \{\mathbb{A}^*_{[4,8)}, \mathbb{A}^*_{[0,2)}, \mathbb{A}^*_{[0,1)}, \mathbb{A}^*_{[1,2)}, \mathbb{A}^*_{[2,3)}\}$. ($\mathbb{A}^*_{[7,8)}$ is excluded from $\Delta\mathbb{P}_2$ since it is cached from $\mathbb{A}_1$ for $\mathbb{W}_1$.) Here, $\|\mathbb{P}_2\|_1 = \|\mathbb{P}_2 \cup \Delta\mathbb{P}_2\|_1 = 2$. Adding any other nodes from the tree to $\Delta\mathbb{P}_2$ will increase the number of nodes in $\mathbb{A} \cup \Delta\mathbb{P}$ that are on the same path of the tree from 2 to 3 or 4, or in other words, $\|\mathbb{P}_2 \cup \Delta\mathbb{P}_2\|_1$ might increase. Instead of any node in $\Delta\mathbb{P}_2$ we could obtain its children nodes, however, our algorithm prefers nodes at the higher layers of the tree, since they are more likely to be reused by other modules. Note that $\Delta\mathbb{P}_2$ does not only consist of disjoint query predicates. For example, $\mathbb{A}^*_{[0,2)}$ and $\mathbb{A}^*_{[0,1)}$ overlap.

6 RELAX PRIVACY MECHANISM

When exploring a database, a data analyst may first ask a series of workloads at a low accuracy (spending ϵ_1), and then re-query the most interesting workloads at a higher accuracy (spending $\epsilon_2 > \epsilon_1$). The cumulative privacy budget spent by the MMM will be $\epsilon_1 + \epsilon_2$ due to sequential composition. The goal of the Relax Privacy module is to spend less privacy budget than MMM on such repeated workloads with higher accuracy requirements.

Koufogiannis et al. [15] refine a noisy response at a smaller ϵ_1, to a more accurate response at a larger ϵ_2, using only a privacy cost of $\epsilon_2 - \epsilon_1$ [15, 24]. However, their framework only operates with the

Algorithm 3 Relax Privacy (RP) (Section 6)

```
1:  function answerWorkload($C, \mathbf{A}, \mathbf{W}, \mathbb{x}$)
2:      $\boldsymbol{\eta}_o \leftarrow \tilde{\mathrm{y}}_o - \mathbb{A}_o\mathbb{x}$   ▷ Old noise vector for $\mathbb{A}_o$.
3:      $\boldsymbol{\eta} \leftarrow$ lapNoiseDown($\boldsymbol{\eta}_o, b_o, b$)   ▷ Koufogiannis et al. [15]
4:      $\tilde{\mathrm{y}} \leftarrow \mathbb{A}_o\mathbb{x} + \boldsymbol{\eta}$   ▷ New noisy responses to $\mathbb{A}_o$
5:      Update cache $C_{\mathbb{A}^*}$ with ($\mathbb{A}_o, b, \tilde{\mathrm{y}}, t$=current time)
6:      $\tilde{\mathrm{y}}' \leftarrow \tilde{\mathrm{y}}$ for $\mathbb{A} \subseteq \mathbb{A}_o$   ▷ New noisy responses to $\mathbb{A}$
7:      return $\mathbf{WA}^+\tilde{y}'$

8:  function estimatePrivacyBudget($C, \mathbf{A}, \mathbf{W}, \alpha, \beta$)
9:      $b \leftarrow$ MMM.estimatePrivacyBudget( $C = \emptyset, \mathbf{A}, \mathbf{W}, \alpha, \beta$)
10:     $C \leftarrow \{\cdots(\mathbb{A}_t, \tilde{\mathrm{y}}_t, b_t)\}$   ▷ Group queries in $C$ by timestamp.
11:     $S_{RP} \leftarrow \mathbb{A}_j \in C_{t=j} | \mathbb{A}_j \supseteq \mathbb{A}$   ▷ Keep only those $\mathbb{A}_t$ that contain $\mathbb{A}$
12:     if $S_{RP} = \emptyset$ then
13:         return "RP cannot run for this input $\mathbb{A}$."
14:     $o = \arg\min_j \quad \epsilon_{RP,j} = \frac{\|\mathbb{A}_j\|_1}{b} - \frac{\|\mathbb{A}_j\|_1}{b_j}$   ▷ $\mathbb{A}_o$ has the lowest RP cost
15:     return $b_o, \quad \epsilon_{RP,o}$   ▷ Cached noise parameter, RP cost for $\mathbb{A}_o$
```

simple Laplace mechanism. Thus, we achieve the aforementioned goal by closely integrating their framework [15] with the matrix mechanism and our DP cache.

6.1 Estimate Privacy Budget Interface

We first describe the estimatePrivacyBudget interface for RP, and as with the MMM, it estimates the privacy budget required by the RP mechanism. The privacy budget required for the RP mechanism is defined as the difference between the new or target privacy budget for the output strategy noisy responses $\tilde{\mathrm{y}}$ to meet the accuracy guarantees, and the old or cached privacy budget (ϵ_C) that cached responses to $\mathbb{A}$ were obtained at. The target noise parameter is the noise parameter required by the cacheless MM to achieve an (α, β)-accuracy guarantee for $\mathbf{W}, \mathbf{A}$. It can be obtained by running the estimatePrivacyBudget of MMM with an empty cache (line 9). Then the main challenge of this interface is to choose which past strategy entries should be relaxed by the RP mechanism, based on the smallest RP cost as defined above.

Each strategy query $\mathbb{a} \in \mathbb{A}$ may be cached at a different timestamp. Relaxing each such set of cache entries across different timestamps, through sequential composition, requires summing over the RP cost for each set, and can thus be very costly. For simplicity, we design the RP mechanism to relax the entirety of a past strategy matrix, rather than picking and choosing strategy entries across different timestamps. Our RP cache lookup condition groups cache entries by their timestamps to form cached strategy matrices (Line 10), and identifies all candidate matrices that include the entire input strategy (Line 11). The inclusion condition (instead of an equality) allows proactively fetched strategy entries to be relaxed, at no additional cost to relaxing $\mathbb{A}_j$. If answering $\mathbb{A}$ using the cache requires: (1) composing cache entries spanning multiple timestamps, or (2) composing cache entries at one timestamp and paid (freshly noised) strategy queries at the current timestamp, then the RP cost estimation interface simply returns nothing (Line 12) and *CacheDP* will instead use another module.

Example 6.1. Suppose that the workloads shown in Figure 1 have been asked in the past at α_1, and have been answered through MMM, as discussed in Example 5.2. Now $\mathbb{W}_3 = \{[3, 8)\}$ is asked at $\alpha_3 < \alpha_1$. We have $\mathbb{A}_3 = \{[3, 4), [4, 8)\}$. Thus $\mathbb{A}_3 \subset \mathbb{P}_2 \cup \Delta\mathbb{P}_2$, and the RP module relaxes all of $\mathbb{A}_{3,RP} = \mathbb{P}_2 \cup \Delta\mathbb{P}_2$ from α_1 to α_3.

For each candidate cached strategy matrix $\mathbb{A}_j$, we compute the RP cost to relax its cached noisy response vector $\tilde{\mathrm{y}}_j$ from b_j to the new target b as $\epsilon_{RP,j}$. Lastly, the RP module chooses to relax the candidate past strategy $\mathbb{A}_j$ with the minimum RP cost (Line 14). For the chosen cached strategy matrix $\mathbb{A}_o$, we return the cached noise parameter b_o and the RP cost $\epsilon_{RP,o}$.

6.2 Answer Workload Interface

The RP answerWorkload interface is a straightforward application of Koufogiannis et al.'s noise down module. We first compute the Laplace noise vector used in the past $\boldsymbol{\eta}_o$, by subtracting the ground truth for the cached old strategy $\mathbb{A}_o\mathbb{x}$ from the cached noisy response $\tilde{y}_o$ (line 2). We can now supply Koufogiannis et al.'s noise down algorithm with the old noise vector $\boldsymbol{\eta}_o$, the cached noise parameter b_o, and the target noise parameter b. This algorithm draws noise from a correlated noise distribution, and outputs a new, more accurate noise vector at noise parameter b (line 3) [15, Algorithm 1]. We can simply compute the new noisy response vector to $\tilde{y}_o$ using the ground truth and the new noise vector (line 4). We then update the cache with the new, more accurate noisy responses, which can be used to answer future strategy queries (line 5). Finally, we do not need the noisy strategy responses to $\mathbb{A}_o - \mathbb{A}$ to answer the data analyst's workload, and so we filter them out to simply obtain new noisy responses $\tilde{y}'$ to $\mathbb{A}$ (line 6). We use $\tilde{y}'$ to compute the workload response and return it to the analyst (line 7).

7 MULTIPLE ATTRIBUTE WORKLOADS

We extend *CacheDP* to work over queries with multiple attributes. We define a single data vector $\mathbb{x}$ over dom$(\mathcal{R})$ as the cross product of d single-attribute domain vectors. It represents the frequency of records for each value of a marginal over all attributes. However, $|\mathbb{x}|$ and thus $|C|$ could be very large due to the cross product.

We observe that not all attributes may be referenced by analysts in their workloads. Suppose that each workload includes marginals over a set of attributes $S_{\mathcal{A}} \in R$. That is, each marginal $\mathbb{w} \in \mathbb{W}$ includes $|S_{\mathcal{A}}| = k \leq d$ RCQs, with one RCQ over each attribute ($\mathbb{w} = \prod_j^k \mathbb{w}_j$). These workloads would share a common, smaller domain and hence a data vector $\mathbb{x}_{S_{\mathcal{A}}}$. Similarly, instead of creating a large cache, we create a set of smaller caches, with one cache $C_{S_{\mathcal{A}}}$ for each unique combination of attributes $S_{\mathcal{A}}$ encountered in a workload sequence. Cache entries can thus be reused across workloads that span the same set of attributes. The entries of each smaller cache $C_{S_{\mathcal{A}}}$ are indexed by its associated domain vector $\mathbb{x}_{S_{\mathcal{A}}}$.

Our cache-aware MMM, SE and RP modules can be extended trivially to the multi-attribute case, since these modules would simply operate on the larger domain vector. However, in order to generate $\mathbb{A}$, the ST module relies on a k-ary strategy tree, corresponding to $\mathbb{A}^*$ for the single-attribute case. Thus, we extend the ST and PQ modules by defining this global strategy tree using marginals over multiple attributes. Our extended ST and PQ modules serve as a proof-of-concept that other modules can be extended for other problem domains. We detail the multi-attribute strategy tree generation in our full paper [18]. We also discuss handling complex SQL queries such as joins, in its future work section.

Dataset	Adult [13]	Taxi [2]	Planes [6, 23]
Size	48842×14	1028527×19	$500,000 \times 12$
Tasks (Attributes)	BFS (Age) DFS (Country)	BFS (Lat, Long) DFS (Lat, Long)	IDEBench (8 out of 12)

Table 2: Datasets, their sizes, and associated tasks, with the attributes or number of attributes used in each task.

8 EVALUATION

We conduct a thorough experimental evaluation of *CacheDP*. We focus on our primary goal, namely, reducing the cumulative privacy budget of interactive workload sequences over baseline solutions (Section 8.2.1), while still meeting the accuracy requirements (Section 8.2.2) and incurring low overheads (Section 8.2.3). We assess how often each module is used, and quantify its impact on the privacy budget, through our ablation study in Section 8.3.

8.1 Experimental Setup

8.1.1 Baseline Solutions. We consider a number of baseline, accuracy-aware solutions from the literature to compare with *CacheDP*.

APEx [9]: *APEx* is a state-of-the-art accuracy-aware interactive DP query engine. *APEx* treats all workload queries separately and has no cache of previous responses.

APEx with cache: We simulate *APEx* with a naive cache of all past workloads and their responses. If a client repeats a workload asked in the past by any client, with the same or a lower accuracy requirement, we do not count its privacy budget towards the cumulative budget spent by *APEx with cache*.

Pioneer [24]: *Pioneer* is a DP query engine that incorporates a cache of previous noisy responses to save the privacy budget on future queries. Since *Pioneer* can only answer single range queries, we decompose all workloads into single queries for our evaluation, and let *Pioneer* answer them sequentially.

8.1.2 Datasets and Tasks. In Table 2, we outline the datasets used and the tasks that each dataset is used in. A common data exploration task involves traversing a decomposition tree over the domain [31]. We construct our workloads through either a breadth-first search (BFS) or a depth-first search (DFS); both of these tasks are executed over a single attribute or a pair of correlated attributes (Latitude and Longitude from the Taxi dataset). We also replicate the evaluation of *Pioneer* [24] through randomized single range queries (RRQ) over a single attribute of a synthetic dataset. We use Eichman et al.'s benchmarking tool, namely IDEBench [6], to construct a sequence of interactive multi-attribute workloads.

We model multiple data analysts querying each system, as clients. We run the BFS, DFS and IDEBench tasks with multiple clients. We schedule the clients' interactions with each system by randomly sampling clients, with replacement, from the set of clients until no queries remain. A client chooses the accuracy requirements and task parameters for each run of the experiment independently and at random; we detail these choices in the full paper [18].

8.2 End-to-end Comparison

8.2.1 Privacy Budget Comparison. We repeat each interactive exploration task $N = 100$ times, and we compute the average cumulative privacy budget for our solution *CacheDP* and baselines (*APEx*, *APEx with cache*, *Pioneer*) over all N experiment runs. We plot the mean and 95% confidence intervals in Figure 3. We have two hypotheses:

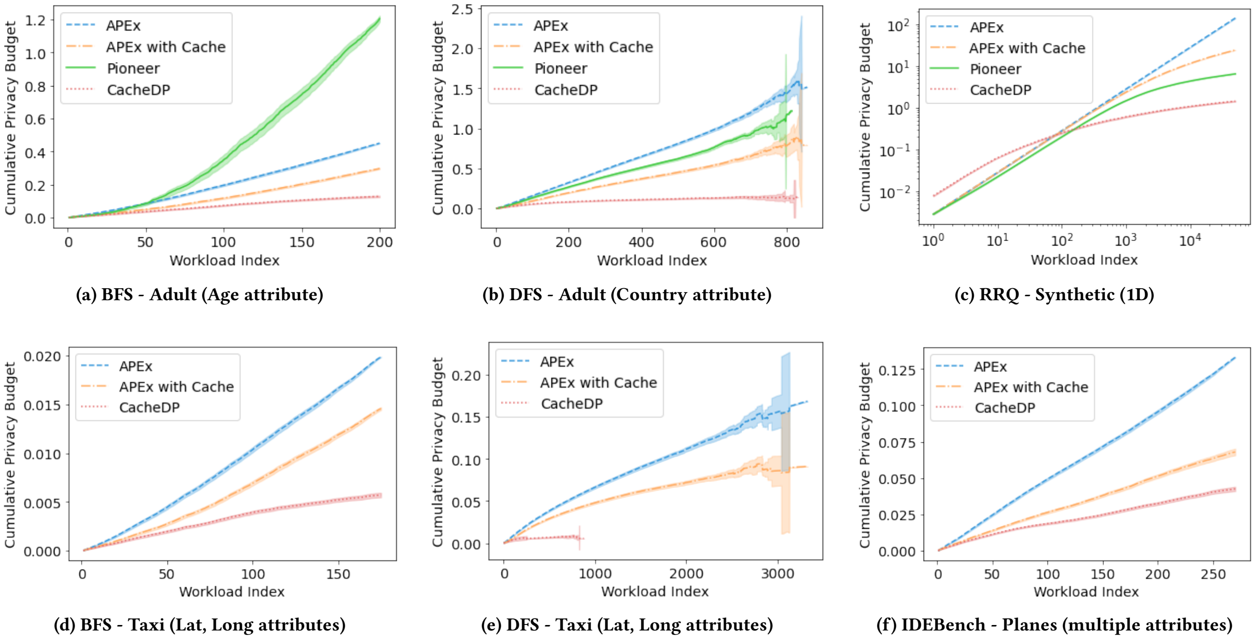
H1 The baselines arranged in order of increasing cumulative privacy budget should be: *APEx*, *APEx with cache*, Pioneer.
  H1.1 *Pioneer* should outperform *APEx with cache*, since *Pioneer* saves privacy budget over any related workloads, whereas *APEx with cache* only saves privacy budget over repeated workloads.
  H1.2 Baselines with a cache (*APEx with cache*, Pioneer) should outperform the baseline without a cache (*APEx*).
H2 *CacheDP* should outperform all baselines.

First, we observe that hypothesis H1 holds for all tasks, other than the single-attribute BFS and DFS tasks (Figures 3a, 3b). Since we decompose each workload into multiple single range queries for Pioneer, this sequential composition causes it to perform worse than *APEx* without a cache in the BFS task, and thus hypothesis H1.1 is violated. For the same reason, *APEx with cache* outperforms Pioneer for the single-attribute DFS task, and so, hypothesis H1.2 is violated. Though, we note that hypothesis H1.2 holds for the RRQ task (Figure 3c). Our Pioneer implementation replicates a similar privacy budget trendline to the original paper [24, Figure 15].

Hypothesis H2 holds for all tasks, and the cumulative privacy budget spent by *CacheDP* scales slower per query, by *at least* a constant factor, over all graphs. We note that in the RRQ task (Figure 3c), *CacheDP* spends more privacy budget *upfront* than the other systems, since these systems use the simpler Laplace Mechanism, which is optimal for single range queries over our underlying Matrix Mechanism. However, any upfront privacy budget spent by *CacheDP* is used to fill the cache, which yields budget savings over a large number of workloads, as *CacheDP* requires an order of magnitude less cumulative privacy budget than the best baseline (*Pioneer*). We observe that even in the computationally intensive tasks due to larger data vectors for two attributes (Figures 3d, 3e) and multiple attributes (Figure 3f), *CacheDP* outperforms the best baseline (*APEx with cache*), by at least a factor of 1.5 for Figure 3f.

For both DFS tasks (Figures 3b, 3e), since each experiment can terminate after a different number of workloads have been run, we observe large confidence intervals for higher workload indices for each system. *CacheDP* simply returns cached responses to a workload if they meet the accuracy requirements, whereas our simulation for *APEx with cache* resamples noisy workload responses and may traverse the tree again in a possibly different path. Relaxed accuracy requirements from different clients can lead to frequent reuse of our cache (Section 8.1.2), and thus, we find that in Figure 3e, *CacheDP* ends the DFS exploration faster than *APEx with cache*.

8.2.2 Accuracy Evaluation. We measured the empirical error of the noisy responses returned by all systems and found that they meet the the clients' (α, β) accuracy requirements. Cached responses used by *CacheDP* commonly exceed the accuracy requirements. Specifically, when *all* strategy responses are free, *CacheDP* will always return the most accurate cached response for each strategy query, even if the current workload has a poorer α.



(a) BFS - Adult (Age attribute)

(b) DFS - Adult (Country attribute)

(c) RRQ - Synthetic (1D)

(d) BFS - Taxi (Lat, Long attributes)

(e) DFS - Taxi (Lat, Long attributes)

(f) IDEBench - Planes (multiple attributes)

Figure 3: Average cumulative privacy budget comparison between CacheDP and baselines (APEx, APEx with cache, Pioneer).

8.2.3 Overhead Evaluation. We compute the following storage and computation overheads for all systems, averaged over all N experiment runs: (1) *cache size* in terms of total number of cache entries at the end of a run, and (2) *workload runtime*, averaged over all workloads in a run. In Table 3, we present these overheads for representative tasks. (Since our simulation for *APEx with cache* only differs from *APEx* by a recalculation of the privacy budget (Section 8.1), the latter has the same runtime as the former.) Our cache size is limited by the number of nodes on our strategy tree, and so for the RRQ task, which has $50k$ workloads, *CacheDP* has a smaller cache size than the baselines. Whereas, in tasks with fewer workloads, such as the IDEBench task, our PQ module inserts more strategy query nodes into the cache, and thus, significantly increases our cache size over the baselines. Nevertheless, since our cache entries only consist of 4 floating points (32B), even a cache with $\approx 25k$ entries would be reasonably small in size ($\approx 800kB$).

In terms of runtime, *CacheDP* only takes a few seconds per workload for single-attribute tasks, such as the DFS task, thereby matching other cached baselines. As the number of attributes increases in the IDEBench task, the cumulative cache size and runtime of *CacheDP* scales linearly. Specifically, IDEBench workloads require computations over a larger data vector that spans many attributes. (We include a graph for these variables and discuss performance optimizations in the extended paper.) Yet, non-optimized *CacheDP* only takes around 6 minutes per workload for the IDEBench task, and performs slightly better than APEx with cache.

8.3 Ablation study

Each of our modules contribute differently to the success of our system across different workloads. We conduct an ablation study in two parts analyzing our modules. First, we analyze the frequencies at which each module is selected to answer a workload, and second, we run a study to quantify the impact of each module on the cumulative privacy budget. We begin with our frequency analysis, noting that a module is chosen to answer a given workload if it is estimated to cost the lowest privacy budget. We only include the MMM, RP, and SE modules in this analysis, since the PQ module is not involved in the cost estimation stage. We present the number of times each module is chosen to answer a workload in each of the BFS, DFS, RRQ and IDEBench tasks, averaged over $N = 100$ runs, in Table 4. If MMM reports $\epsilon = 0$ for a workload, *CacheDP* simply uses MMM to answer the workload using cached responses, and it does not run RP or SE modules. We thus separately record the number of free workloads per task in the first row of Table 4. First, we observe that most workloads for each task are free, indicating that using solely the cached strategy responses, *CacheDP* can successfully answer most workloads for these tasks.

Second, considering all non-free workloads, each of the modules are used the most frequently for at least one task. SE is chosen most frequently for the BFS and DFS tasks, answering 52% and 64% of non-free workloads respectively. Furthermore, for many workloads in these tasks, we observed that MMM had $\epsilon > 0$ cost, but under SE, these workloads became free ($\epsilon = 0$). RP is chosen most frequently for the IDEBench task (57%), whereas MMM is used most frequently for the RRQ task (69%). Thus, we can see that each module plays a role in *CacheDP*'s performance in one or more tasks.

We also run a study to quantify savings in the cumulative privacy budget due to each module. We rerun our single-attribute tasks (BFS, DFS, RRQ) while disabling each of our modules (MMM, SE, RP, PQ) one at a time, and present the cumulative privacy budget consumed by each such configuration in Figure 4. The standard configuration

System	Cache size (entries)		Runtime (s)	
	RRQ	IDEBench	DFS - Adult	IDEBench
APEx (cache)	3118±4	6540±0	2.71±0.02	456±70
Pioneer	3118±4	-	1.6±0.2ms	-
CacheDP	1998±0	23666±1000	2.82±0.2	338±20

Table 3: Cache size and workload runtime comparison.

	BFS	DFS	RRQ	IDEBench
Free	164.8 ± 0.9	591 ± 6	49801 ± 3	216 ± 1
MMM	2.1 ± 0.1	2.25 ± 0.09	**137 ± 3**	15.1 ± 0.2
RP	14.7 ± 0.5	21.1 ± 0.5	1.4 ± 0.1	**31 ± 1**
SE	**18.5 ± 0.6**	**42 ± 1**	60 ± 4	7.8 ± 0.2

Table 4: Average number of times each module was chosen for each task; most frequently chosen modules are in bold.

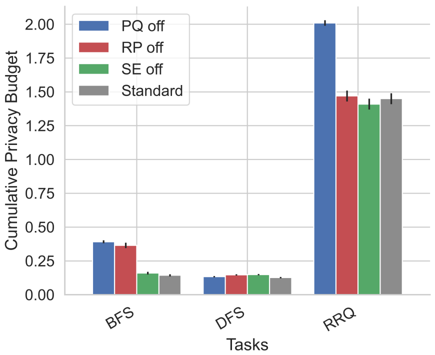


Figure 4: Ablation study over PQ, RP, SE modules of *CacheDP*

consists of all modules turned on. (Turning an effective module off should lead to an increase in the cumulative privacy budget, in comparison to the standard configuration.) First, we observe that the standard configuration performs the best in all three tasks, while considering CI overlaps. Therefore, data analysts need not pick which modules should be turned on in order to answer a workload sequence with the lowest privacy budget. Second, the PQ module significantly lowers the cost for the BFS and RRQ task, proactively fetching all (≈ 12) queries in a BFS workload at the cost of one strategy node. Third, the RP module also lowers the cost for BFS, when the same workload is repeated by other clients.

Fourth, turning the SE module off only contributes to minor differences in the cumulative privacy budget (B_c). However, the reader may expect that turning the SE module off would lead to a higher B_c, since based on the frequency analysis, the SE module is most frequently chosen to answer non-zero workloads for the BFS and DFS tasks. We reconcile this discrepancy with the observation that the SE module provides savings on earlier workloads, through constrained inference, at the cost of a less accurate cache to answer future workloads. In summary, different tasks exploit different modules and the standard configuration incurs the least privacy budget, and thus data analysts need not turn off any modules.

9 RELATED WORK

Constrained inference techniques have been applied in the non-interactive DP setting to improve the accuracy of noisy query responses [25, 31] and in synthetic data generators [10, 19, 26] to infer consistent answers from a data model built through noisy measurement queries. However, these systems do not provide any accuracy guarantee on the inferred responses. If the analyst desires a more accurate response than the synthetic data can offer, no privacy budget remains to improve the query answer [27]. Our work applies DP constrained inference in an interactive setting so that we can spend the privacy budget on queries that the analyst is interested in and meet their accuracy requirements. On the other hand, existing accuracy-aware DP systems for data exploration [9, 21], releasing data [8, 22], or programming frameworks [28, 30] do not exploit historical query answers to save privacy budget on a given query. We design a cache structure and inference engine extending one of these accuracy-aware systems, APEx [9].

Peng et al.'s Pioneer [24] is the most relevant work that uses historical query answers to obtain accurate responses to upcoming single range queries. However, *CacheDP* can handle workloads with multiple queries. Second, it supports multiple, complex DP mechanisms and chooses the mechanism that uses the least privacy budget for each new workload. Third, our PQ module (Section 5.3) proactively fetches certain query responses that can be used later at no additional cost. Finally, *CacheDP* can answer multi-attribute queries through our extended ST module (Section 7).

Our key modules are built on top of prior work (e.g., Li et al.'s Matrix Mechanism [16], Koufogiannis et al.'s Relax Privacy Mechanism [15]), such that existing interactive DP systems that make use of these mechanisms (e.g. PrivateSQL [14], APEx [9]) do not have to make significant changes; these systems can include a relatively light-weight cache structure and cache-aware version of the DP mechanisms. Integrating a structured, reusable cache with these mechanisms has its own technical challenges, such as the Cost Estimation problem (Section 4.3.2), Full Rank Transformation problem (Section 5.1.2), as well as optimally reusing the cache (Section 5.2) and filling it (Section 5.3).

10 CONCLUSION

We build a usable interactive DP query engine, *CacheDP*, that uses a structured DP cache to achieve privacy budget savings commonly seen in the non-interactive model. *CacheDP* supports data analysts in answering data exploration workloads accurately, without requiring them to have any DP knowledge. Our work provides researchers with a methodology to address common challenges while integrating DP mechanisms with a DP cache, such as, cache-aware privacy budget estimation (MMM), filling the cache at a low privacy budget (PQ), and maximizing cache reuse (SE).

ACKNOWLEDGMENTS

We thank NSERC for funding our work through the Postgraduate Scholarship-Doctoral program, grant CRDPJ-534381, and a Discovery Grant. We also thank the Royal Bank of Canada for supporting our work. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

REFERENCES

[1] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 441–459. https://doi.org/10.1145/3132747.3132769

[2] NYC Taxi & Limousine Commission. 2022. *TLC Trip Record Data*. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[3] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. 3574–3583. https://doi.org/10.5555/3294996.3295115

[4] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, Shai Halevi and Tal Rabin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 265–284.

[5] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3–4 (2014), 211–407. https://doi.org/10.1561/0400000042

[6] Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2020. IDEBench: A Benchmark for Interactive Data Exploration. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1555–1569. https://doi.org/10.1145/3318464.3380574

[7] Marco Gaboardi, Michael Hay, and Salil Vadhan. 2019. *A Programming Framework for OpenDP*. https://projects.iq.harvard.edu/opendp

[8] Marco Gaboardi, James Honaker, Gary King, Jack Murtagh, Kobbi Nissim, Jonathan Ullman, and Salil Vadhan. 2016. Psi: a private data sharing interface. (2016). http://arxiv.org/abs/1609.04340

[9] Chang Ge, Xi He, Ihab F. Ilyas, and Ashwin Machanavajjhala. 2019. APEx: Accuracy-Aware Differentially Private Data Exploration. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands). Association for Computing Machinery, New York, NY, USA, 177–194. https://doi.org/10.1145/3299869.3300092

[10] Chang Ge, Shubhankar Mohapatra, Xi He, and Ihab F. Ilyas. 2021. Kamino: Constraint-Aware Differentially Private Data Synthesis. *VLDB* (2021).

[11] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the Accuracy of Differentially Private Histograms through Consistency. *Proceedings of the VLDB Endowment* 3, 1–2 (Sept. 2010), 1021–1032. https://doi.org/10.14778/1920841.1920970

[12] Noah Johnson, Joseph P. Near, and Dawn Song. 2018. Towards Practical Differential Privacy for SQL Queries. *Proceedings of the VLDB Endowment* 11, 5, 526–539. https://doi.org/10.1145/3177732.3177733

[13] Ron Kohavi. 1996. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. https://archive.ics.uci.edu/ml/datasets/adult. In *KDD*, Vol. 96. 202–207.

[14] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: A Differentially Private SQL Query Engine. *Proceedings of the VLDB Endowment* 12, 11 (July 2019), 1371–1384. https://doi.org/10.14778/3342263.3342274

[15] Fragkiskos Koufogiannis, Shuo Han, and George J Pappas. 2016. Gradual Release of Sensitive Data under Differential Privacy. *Journal of Privacy and Confidentiality* 7, 2 (2016), 23–52. https://doi.org/10.29012/jpc.v7i2.649

[16] Chao Li, Gerome Miklau, Michael Hay, Andrew Mcgregor, and Vibhor Rastogi. 2015. The Matrix Mechanism: Optimizing Linear Counting Queries under Differential Privacy. *The VLDB Journal* 24, 6 (Dec. 2015), 757–781. https://doi.org/10.1007/s00778-015-0398-x

[17] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. 2008. Privacy: Theory meets practice on the map. In *2008 IEEE 24th international conference on data engineering*. IEEE, IEEE, Cancun, Mexico, 277–286. https://doi.org/10.1109/ICDE.2008.4497436

[18] Miti Mazmudar, Thomas Humphries, Jiaxiang Liu, Matthew Rafuse, and Xi He. 2022. *Cache Me If You Can: Accuracy-Aware Inference Engine for Differentially Private Data Exploration*. Technical Report. University of Waterloo. https://doi.org/10.48550/arXiv.2211.15732

[19] Ryan McKenna, Daniel Sheldon, and Gerome Miklau. [n.d.]. Graphical-model based estimation and inference for differential privacy. arXiv:1901.09136

[20] Frank McSherry. 2010. Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis. *Commun. ACM* 53, 9 (2010), 89–97. https://doi.org/10.1145/1810891.1810916

[21] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. 2012. GUPT: Privacy Preserving Data Analysis Made Easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 349–360. https://doi.org/10.1145/2213836.2213876

[22] Priyanka Nanayakkara, Johes Bater, Xi He, Jessica Hullman, and Jennie Rogers. 2022. Visualizing Privacy-Utility Trade-Offs in Differentially Private Data Releases. *CoRR* (2022). arXiv:2201.05964

[23] United States Department of Transportation. 2022. Bureau of Transportation Statistics. https://transtats.bts.gov

[24] S. Peng, Y. Yang, Z. Zhang, M. Winslett, and Y. Yu. 2013. Query optimization for differentially private data management systems. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, Brisbane, QLD, Australia, 1093–1104. https://doi.org/10.1109/ICDE.2013.6544900

[25] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Understanding hierarchical methods for differentially private histograms. In *Proceedings of the VLDB Endowment*, Vol. 6. 1954–1965. http://www.vldb.org/pvldb/vol6/p1954-qardaji.pdf

[26] Uthaipon Tantipongpipat, Chris Waites, Digvijay Boob, Amaresh Siva, and Rachel Cummings. 2021. Differentially private synthetic mixed-type data generation for unsupervised learning. *Intelligent Decision Technologies* (2021).

[27] Yuchao Tao, Ryan McKenna, Michael Hay, Ashwin Machanavajjhala, and Gerome Miklau. 2021. Benchmarking Differentially Private Synthetic Data Generation Algorithms. *TPDP* (2021). https://tpdp.journalprivacyconfidentiality.org/2021/papers/NingUQKH21.pdf

[28] Elisabet Lobo Vesga, Alejandro Russo, and Marco Gaboardi. 2019. A Programming Framework for Differential Privacy with Accuracy Concentration Bounds. *CoRR* (2019). arXiv:1909.07918

[29] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. 2020. Differentially Private SQL with Bounded User Contribution. *Proceedings on Privacy Enhancing Technologies* 2020 (2020), 230–250. Issue 2. https://doi.org/10.2478/popets-2020-0025

[30] Yingtai Xiao, Zeyu Ding, Yuxin Wang, Danfeng Zhang, and Daniel Kifer. 2021. Optimizing Fitness-for-Use of Differentially Private Linear Queries. *Proceedings of the VLDB Endowment* 14, 10 (2021), 1730–1742. https://doi.org/10.14778/3467861.3467864

[31] Jun Zhang, Xiaokui Xiao, and Xing Xie. 2016. PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, 155–170. https://doi.org/10.1145/2882903.2882928