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

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

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

Jiaxiang Liu
j632liu@uwaterloo.ca
University of Waterloo

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

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

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.

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) [6, 7] 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 [19], Google [30] and Microsoft [4].

Existing DP deployments [3, 4, 16, 19] mainly consider a non-interactive setting, where the analyst provides all queries in advance. Whereas in interactive DP systems [9, 14, 22, 30], 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 [11] 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. [13] 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 [25]. 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 Mechanism [18] (Section 4), and Koufogiannis et al.'s Relax Privacy mechanism [17] (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.

2 BACKGROUND

We consider a single-table relational schema $\mathcal{R}$ across d attributes: $\mathcal{R}(\mathcal{A}_1, \ldots \mathcal{A}_d)$. 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 $\varkappa$ of length n, where $\varkappa[i]$ is the count of ith value from $dom(\mathcal{R})$ in D. After constructing $\varkappa$, we represent any RCQ as a length-n vector $\varkappa$ with $\varkappa[i] \in \{0, 1\}$ for $i = 1, \ldots, n$. To obtain the ground truth response for a RCQ $\varkappa$, we can simply compute $\varkappa \cdot \varkappa$. 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 $\mathbb{W}\varkappa$.

When we partition the full domain $dom(\mathcal{R})$ into a set of n' disjoint buckets, the data vector $\varkappa$ 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.

Definition 2.1 (ϵ-Differential Privacy (DP) [6]). A randomized mechanism $M : \mathcal{D} \rightarrow O$ satisfies ϵ-DP if for any output sets $O \subseteq O$, 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 [6, 18]). *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*

Table 1: Notation

Notation	Description
$\varkappa, \varkappa, \mathbb{W}, \mathbb{A}$	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 $\mathbf{W}$.
$(\mathbb{Q}, b, \tilde{y}, t)$	a cache entry for a strategy query $\mathbb{Q} \in \mathbb{A}^*$ stored at timestamp t. See Definition 3.1.
$\mathbf{F}, \mathbf{P}$	free strategy matrix, paid strategy matrix

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. [18] 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) [18]). 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: $\mathcal{M}_{\mathbf{A}, \mathcal{L}_b}(\mathbf{W}, \mathbf{x}) = \mathbf{W}\mathbf{A}^+ \mathcal{L}_b(\mathbf{A}, \mathbf{x})$ 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 matrix mechanism post-processes the output of a DP mechanism [7], it also satisfies the same level of privacy guarantee.

Proposition 2.1 ([18]). *If $b \geq \frac{\|\mathbf{A}\|_1}{\epsilon}$, then $\mathcal{M}_{\mathbf{A}, \mathcal{L}_b}$ satisfies ϵ-DP.*

For data analysts who may not be able to choose an appropriate budget for a DP mechanism, we would like to allow them to specify their accuracy requirements for their queries. We consider two popular types of error specification 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 [18] 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 [11] 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. [11] provides a loose bound for the noise parameter in the matrix mechanism to achieve an (α, β)-worst error bound.

Theorem 2.2 ([11]). *The matrix mechanism $\mathcal{M}_{\mathbf{A}, \mathcal{L}_b}$ satisfies the (α, β)-worst error bound, if*

$$b \leq b_L = \frac{\alpha\sqrt{\beta/2}}{\|\mathbf{W}\mathbf{A}^+\|_F} \tag{4}$$
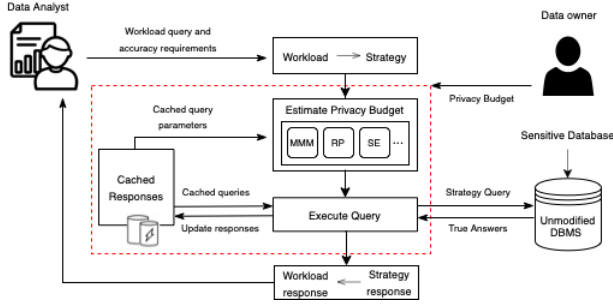
Figure 1: System diagram.

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 $\frac{\|A\|_1}{b_L}$. To minimize the privacy cost, Ge et al. [11] 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 $\|WA^+Lap(b)^l\|_\infty$ (i.e., if the sampled error exceeds α with a probability $\leq \beta$).

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 system architecture is given in Figure 1. 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 $\tilde{z}$, 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 $[(\mathbb{W}_1, \alpha_1, \beta_1, \tilde{z}_1), \ldots, (\mathbb{W}_t, \alpha_t, \beta_t, \tilde{z}_t)]$. When a new workload $(\mathbb{W}_{t+1}, \alpha_{t+1}, \beta_{t+1})$ comes in, the system first infers a response $\tilde{z}'_{t+1}$ from the cache and its error bound α'_{t+1}. If its error bound is worse than the accuracy

requirement, i.e., $\alpha'_{t+1} \geq \alpha_{t+1}$, then additional privacy budget ϵ_{t+1} needs to be spent to improve $\tilde{z}'_{t+1}$ to $\tilde{z}_{t+1}$. 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 [25], 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 [25] 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 (MM) 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 parameter.

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 $\mathbb{q} \in \mathbb{A}^*$, with its noisy response, its noise parameter, and the timestamp.
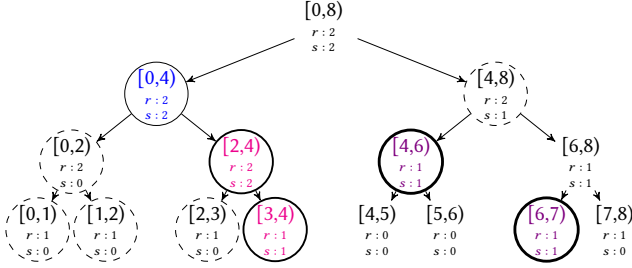
Definition 3.1 (Cache Structure). Given a global strategy matrix $\mathbb{A}^*$ over the full domain $dom(\mathcal{R})$, a cache for differentially private counting queries is defined as

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

where b and $\tilde{y}$ are the latest noise parameter and noisy response for the strategy query $\mathbb{q}$, and t is the time stamp for the latest update of $\mathbb{q}$. At beginning, all entries are initialized as $(\mathbb{q}, -, -, 0)$, 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 [18] with an expected worst error of $O(\log^3 n)$, where n is the domain size. Figure 2 shows the global strategy matrix as a binary tree decomposition of a small integer domain $[0, 8)$.

Figure 2: A global strategy $\mathbb{A}^*$ in a binary tree decomposition for an integer domain $[0,8)$. Workload queries include $\mathbb{W}_1 = \{[0,7)\}$, $\mathbb{W}_2 = \{[2,6), [3,7)\}$. Nodes present only in strategy $\mathbb{A}_1$ are shown in blue text, nodes only in $\mathbb{A}_2$ are in magenta text, nodes in both $\mathbb{A}_1$ and $\mathbb{A}_2$ are shown in purple text. The dashed nodes are output by the Proactive Querying (PQ) module (Section 5.3), for $\mathbb{A}_2$; the value annotations for r and s refer to Algorithm 6.

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 (FT). Prior work [18] uses the global strategy $\mathbb{A}^*$, which has a high $\|\mathbb{A}^*\|_1$. Given an input $\mathbb{W}$, the SG selects a basic instant strategy $\mathbb{A} \subseteq \mathbb{A}^*$, with a low $\|\mathbb{A}^*\|_1$, among other criteria. Though the cache $C_{\mathbb{A}^*}$ is structured based on $\mathbb{A}^*$, the cache is not searched while generating $\mathbb{A}$. We present two example workloads and the instant strategies generated for these workloads next.

Example 3.1. In Figure 2, 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 $\mathbf{W}$, $\mathbf{A}$, 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.

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, α, β) 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 ϵ_i
11:    **if** $\epsilon_i + B_c \geq \mathcal{B}$ **then**
12:       Answering Q satisfying (α, β) 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

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). These two cache-aware DP mechanisms commonly use the ST module to transform an input $\mathbb{W}, \mathbb{x}$ to $(\mathbf{W}, \mathbf{A}, \mathbf{x})$. Each cache-aware DP mechanism implements two interfaces (similar to APEx [11]) using the mapped representations $\mathbf{A}, \mathbf{W}, \mathbf{x}$, 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 $C_{\mathbb{A}^*}$.

- 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. [13]. 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)

In this section, we focus on our core cache-aware DP mechanism, namely the Modified Matrix Mechanism (MMM). We would like to answer a workload $\mathbf{W}$ with an (α, β)-accuracy requirement using a given cache $C_{\mathbf{A}^*}$ and an instant strategy $\mathbf{A} \subseteq \mathbf{A}^*$, while minimizing the privacy cost. We will first provide intuition for the design of this mechanism. Then, we will describe the first interface ANSWER-WORKLOAD that answers a workload $\mathbf{W}$ using the instant strategy $\mathbf{A}$ with the best set of parameters derived from the second interface ESTIMATEPRIVACYBUDGET. We then present how the the second interface arrives at an optimal privacy budget.

4.1 MMM Overview

The cacheless matrix mechanism (Definition 2.2) perturbs the ground truth response to the strategy, that is $\mathbf{A}\mathbf{x}$, with the noise vector freshly drawn from $Lap(b)^{|\mathbf{A}|}$ to obtain $\tilde{\mathbf{y}} = \mathbf{A}\mathbf{x} + Lap(b)^{|\mathbf{A}|}$. An input workload is then answered using $\mathbf{W}\mathbf{A}^+\tilde{\mathbf{y}}$. As we discussed in the background, in an accuracy-aware DP system such as APEx [11], 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 <u>accurately cached</u> 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 <u>free strategy matrix</u> and the <u>paid strategy matrix</u> 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

---

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:      $\tilde{\mathbf{y}}_{\mathbf{P}} \leftarrow \mathbf{P}\mathbf{x} + Lap(b_{\mathbf{P}})^{|\mathbf{P}|}$ ▷ 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:      $\tilde{\mathbf{y}}_{\mathbf{F}} \leftarrow [(w, b, \tilde{y}, t) \in C | \mathbf{w} \in \mathbf{F}]$ ▷ 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** $\mathbf{W}\mathbf{A}^+\tilde{\mathbf{y}}, \epsilon$

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 $b_L = \frac{\alpha\sqrt{\beta/2}}{\|\mathbf{W}\mathbf{A}^+\|_F}$ ▷ Theorem 2.2 (without cache)
12:      $\mathbf{b} \leftarrow [(w, b, \tilde{y}, t) \in C \mid \mathbf{w} \in \mathbf{A} \cap C, b > b_L] \cup [b_L]$
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** ($\mathbf{F}, \mathbf{P}, b_{\mathbf{P}}, \frac{\|\mathbf{P}\|_1}{b_{\mathbf{P}}}$)

17: **function** CHECKACCURACY($b_{\mathbf{P}}, C, \mathbf{A}, \mathbf{W}, \alpha, \beta$)
18:      $(\mathbf{F}, \mathbf{b}_{\mathbf{F}}) \leftarrow [(w, b, \tilde{y}, t) \in C \mid \mathbf{w} \in \mathbf{A} \cap C, b < b_{\mathbf{P}}]$ and $\mathbf{P} \leftarrow \mathbf{A} - \mathbf{F}$
19:      Sample size $N = 10000$ and failure counter $n_f = 0$
20:      **for** $i = 1, \ldots, N$ **do**
21:          n_f++ if $\|\mathbf{W}\mathbf{A}^+ Lap(\mathbf{b}_{\mathbf{F}} \| b_{\mathbf{P}})\|_{\infty} > \alpha$
22:      $\beta_e = n_f/N$, $p = \beta/100$
23:      $\delta\beta = z_{1-p/2}\sqrt{\beta_e(1-\beta_e)/N}$
24:      **return** $(\beta_e + \delta\beta + p/2) < \beta$

---

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 ESTIMATEPRIVA-CYBUDGET 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 $\mathbf{W}\mathbf{A}^+\tilde{\mathbf{y}}$, and its privacy cost ϵ.

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: $\mathbf{b} = \mathbf{b_F} || \mathbf{b_P}$, where $\mathbf{b_F}$ is a vector of noise parameters for the cached entries in $\mathbf{F}$ with length $|\mathbf{F}|$ and $\mathbf{b_P}$ is a vector of the same value b_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 $\mathbf{b} = \mathbf{b_F}||\mathbf{b_P}$, the error to a workload $\mathbf{W}$ using the ANSWERWORKLOAD interface of MMM (Algorithm 2) is

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

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

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

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 $\mathbf{A} - \mathbf{F}$.

(2) Given $\mathbf{P}$ and b_P, the privacy budget paid by MMM is given by $\epsilon = \|\mathbf{P}\|_1 / b_P = \|\mathbf{A} - \mathbf{F}\|_1 / b_P$. To minimize this privacy budget, what is the maximum noise parameter value b_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_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 3. 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 $\|\mathbf{W}\mathbf{A}^+ diag(\mathbf{b})\|_F^2 = 300$ 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 $\mathbf{b} = [15, b_P, b_P]$, and the expected square error is $\|\mathbf{W}\mathbf{A}^+ diag(\mathbf{b})\|_F^2 = 15^2 + 2b_P^2$. To achieve the same or a smaller error than the cacheless MM, we need to set $b_P \leq \sqrt{(300 - 15^2)/2} \approx 6.12$ for the remaining entries in the strategy. This tighter noise parameter b_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.

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_c \\ b \\ b \end{bmatrix}, \mathbf{x}_1 = \begin{bmatrix} \varkappa[0,4) \\ \varkappa[4,6) \\ \varkappa[6,7) \end{bmatrix}$$

Figure 3: 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.**

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 $b_P \in [b_L, b_T]$ that minimizes the paid privacy budget $\epsilon = \frac{\|\mathbf{P}\|_1}{b_P}$ subject to accuracy requirement:
$$\|\mathbf{W}\mathbf{A}^+ diag(\mathbf{b_F}||\mathbf{b_P})\|_F^2 \leq \alpha^2 \text{ or }$$
$$\Pr[\|\mathbf{W}\mathbf{A}^+ Lap(\mathbf{b_F}||\mathbf{b_P})\|_\infty \geq \alpha] \leq \beta.$$

In this optimization problem, the lower bound for b_P is the loose bound for the cacheless MM (Equation (4)), and the upper bound b_T is $\frac{\|\mathbf{A}\|_1}{\epsilon_\perp}$, 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 $b_P \in [b_L, b_T]$, and check whether every possible pair of $(\mathbf{F}, b_P)$ 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 $O\left(2^{|\mathbf{A} \cap C|} \cdot \log_2(|[b_L, b_T]|)\right)$ if we apply binary search within $[b_L, b_T]$. Hence, we need another way to efficiently determine optimal values for $(\mathbf{F}, b_P)$.

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

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

and formalize a simplified optimization problem.

Simplified CE problem: Given a cache C and an instant strategy matrix $\mathbf{A}$, determine $b_P \in [b_L, b_\perp]$ (and $\mathbf{F} = \mathbf{F}_{b_P}, \mathbf{P} = \mathbf{A} - \mathbf{F}$) that minimizes the paid privacy budget $\epsilon = \frac{\|\mathbf{P}\|_1}{b_P}$ subject to:
$$\|\mathbf{W}\mathbf{A}^+ diag(\mathbf{b_F}||\mathbf{b_P})\|_F^2 \leq \alpha^2 \text{ or }$$
$$\Pr[\|\mathbf{W}\mathbf{A}^+ Lap(\mathbf{b_F}||\mathbf{b_P})\|_\infty \geq \alpha] \leq \beta.$$

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. We visualize our searches through the cached noise parameters in
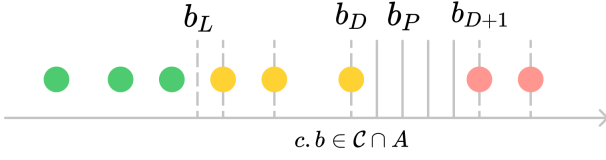
Figure 4: An example of cached noise parameters $c.b \in A \cap C$ (dots) and the discrete (dashed lines) and continuous (full lines) binary searches through these parameters. Parameters in green are accurate enough. The discrete search scans over $c.b \geq b_L$ and outputs b_D; the free matrix includes all cache entries in green and yellow. The continuous search scans over the interval $[b_D, b_{D+1}]$ and identifies an optimal b_P.

Figure 4. First, we setup the upper and lower bounds for the noise parameter b_P for the simplified CE problem (Lines 10-11).

Step 1: Discrete search for b_P. We first search b_P from the existing noise parameters in the cached strategy rows $A \cap C$ 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_P achieves (α, β)-accuracy requirement, we run the function CHECKACCURACY , defined in Algorithm 2 . This function first places all the cached entries with noise parameter smaller than b_P into $\mathbf{F}$ and the remaining entries of the strategy into $\mathbf{P}$ (Line 18) . Then it runs an MC simulation (Lines 19-24) 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_P achieves (α, β)-accuracy guarantee. This MC simulation differs from a traditional one [11] 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_P in a continuous space. We observe that we may further increase b_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}. The continuous search is depicted through full lines in Figure 4. This search outputs a noise parameter b_P. Finally, this function returns b_P, the privacy budget $\epsilon = \frac{\|\mathbf{P}\|_1}{b_P}$, 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 $n_c = |A \cap C|$. 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).

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 $\mathbb{W}$ [18], 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 [13]. 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 $\mathbf{A}$ [18, Section 4]. We present a full-rank transform (FRT) component to address this last requirement. Thus the ST module consists of two components: the strategy generator, and the full-rank transform, run sequentially.

5.1.1 Strategy Generator. Our global strategy $\mathbb{A}^*$ is a k-ary tree over the full domain $dom(\mathcal{R})$, hence, it supports all possible counting queries on the full domain. A baseline instant strategy $\mathbb{A}$ just uses the full global strategy matrix ($\mathbb{A} = \mathbb{A}^*$), thus satisfying the first requirement. To answer the first workload, we obtain the noisy strategy responses for all nodes on the tree, thereby fully populating the cache and reusing the cached noisy responses for future workloads. However, this instant strategy 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. Moreover, this high upfront cost may not be amortized across future workload queries, for example, if the future queries do not require many nodes on this tree. Future workload queries may also have higher accuracy requirements, and we would thus need to re-sample noisy responses to the entire tree again, with a lower noise parameter.

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 [13]. 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.

Algorithm 3 Strategy Transformer (ST)

1: **function** DECOMPOSEWORKLOAD($\mathbb{w}$, node v)
2:     **if** v.query == $\mathbb{w}$ **then return** v
3:     $\mathbb{o} \leftarrow \emptyset$
4:     **if** v has children **then**
5:         **for** child c of node v **do**
6:             $\mathbb{w}_c \leftarrow$ OVERLAPPINGRCQ($\mathbb{w}$, node c.query)
7:             **if** $\mathbb{w}_c \neq \emptyset$ **then**
8:                 $\mathbb{o} \leftarrow \mathbb{o} \cup$ DECOMPOSESTRATEGY($\mathbb{w}_c$, node c)
9:     **return** $\mathbb{o}$

Example 5.1. We continue with Example 3.1 shown in Figure 2, 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 (Line 2). 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, $\|\mathbb{A}_1\|_1 = 1 < \|\mathbb{A}^*\|_1$.

The second workload $\mathbb{W}_2$ has two queries with range predicates ($[2, 6)$, $[3, 7)$). The first workload query predicate requires the strategy nodes $\mathbb{A}^*_{[2,4)}$ and $\mathbb{A}^*_{[4,6)}$, whereas the second query requires the following three nodes: $\mathbb{A}^*_{[3,4)}, \mathbb{A}^*_{[4,6)}$ and $\mathbb{A}^*_{[6,7)}$. Hence, the second instant strategy $\mathbb{A}_2$ is a set of all of these strategy nodes.

The global strategy has an L_1 norm $\|\mathbb{A}^*\|_1 = 4$. The matrix forms of $\mathbb{A}_1$ and $\mathbb{A}_2$ can be generated as shown in Example 5.2. Both strategy matrices improve over the global strategy in terms of their L_1 norms: $\mathbb{A}_1 = 1 < \|\mathbb{A}^*\|_1$ and $\mathbb{A}_2 = 2 < \|\mathbb{A}^*\|_1$. 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. □

We formalize our strategy generation algorithm in the recursive function WORKLOADDECOMPOSE given in Algorithm 4. This function takes as input a single workload query range interval $\mathbb{w}$ and a node v on the tree $\mathcal{T}$. It first checks if the input predicate matches the range interval for the node v. If it does, it returns that range interval (Line 2). Otherwise, for each child c of node v, it computes the overlap $\mathbb{w}_c$ of the range interval of that child with the interval $\mathbb{w}$ (Line 6). For instance, the overlap of the range interval $[2, 6)$ with $[0, 4)$ is given by $[(max)\{(0, 2)\}, min\{(4, 6)\}) = [2, 4)$. For each child with a non-null range interval overlap, the function is called recursively with that overlap $\mathbb{w}_c$ (Line 8). This function is called with the root of the tree $\mathcal{T}$, as the second argument, and returns with the decomposition of $\mathbb{w}$ over all child nodes in the tree ($\mathbb{o}$). It is run for each workload RCQ $\mathbb{w} \in \mathbb{W}$, and $\mathbb{A}$ simply includes the union of each workload decomposition.

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

---

Algorithm 4 Full-rank transformer (Section 5.1.2)

1: **function** GETTRANSFORMATIONMATRIX($\mathbb{A}$)
2:     $\mathbb{T} = \{\mathbb{A}[0]\}$
3:     **for** i in range$(1, |\mathbb{A}|)$ and $\mathbb{A}[i] \notin \mathbb{T}$ **do**
4:         **if** $\mathbb{t} \cdot \mathbb{A}[i] = 0$ for all $\mathbb{t} \in \mathbb{T}$ **then**
5:             $\mathbb{T} \leftarrow \mathbb{T} \cup \{\mathbb{A}[i]\}$ ▹ Add a disjoint bucket $\mathbb{A}[i]$
6:         **else**
7:             $\mathbb{T}' \leftarrow \{\mathbb{t} \in \mathbb{T} \mid \mathbb{t} \cdot \mathbb{A}[i] = 0\}$
8:             **for** $\mathbb{t}$ in $\mathbb{T}$ and $\mathbb{t} \cdot \mathbb{A}[i] \neq 0$ **do**
9:                 $\mathbb{T}' \leftarrow \mathbb{T}' \cup (\mathbb{t} \cap \mathbb{A}[i]) \cup (\mathbb{t} - \mathbb{A}[i])$
10:             $\mathbb{T}' \leftarrow \mathbb{T}' \cup (\mathbb{A}[i] - \sum_{\mathbb{t} \in \mathbb{T}' \land \mathbb{t} \cdot \mathbb{A}[i] \neq 0} \mathbb{t})$
11:         $\mathbb{T} = \mathbb{T}'$
12:     **return** $\mathbb{T}$

13: **function** TRANSFORMSTRATEGY($\mathbb{A}$)
14:     $\mathbb{T} \leftarrow$ GETTRANSFORMATIONMATRIX($\mathbb{A}$)
15:     Initialize $\mathbf{A}$ as a $|\mathbb{A}| \times |\mathbb{T}|$ zero-valued matrix
16:     **for** i in range$(|\mathbb{A}|)$ and j in range$(|\mathbb{T}|)$ **do**
17:         **if** $\mathbb{T}[j] \subseteq \mathbb{A}[i]$ **then**
18:             Set $\mathbf{A}[i, j] = 1$ ▹ Bucket j is contained in query j
19:     **return** $\mathbf{A}, \mathbb{T}$

---

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, $\mathbb{T}[j, i] = 0$.

We elaborate exactly how we create a transformation matrix $\mathbb{T}$ to support strategy $\mathbb{A}$, as presented in GETTRANSFORMATIONMATRIX in Algorithm 4. To create $\mathbb{T}$, GETTRANSFORMATIONMATRIX starts with the first row of $\mathbb{A}$ (line 2), then iterates through the remaining rows (line 3) updating the transformation matrix $\mathbb{T}$ as needed. If row i is disjoint from all buckets, we simply add this row as a new bucket (line 4). Otherwise, we construct a new bucket matrix $\mathbb{T}'$. To do this, we first copy all rows of $\mathbb{T}$ that do not intersect with the current row of $\mathbb{A}$ (line 7). Then, for the buckets that do intersect, we remove the intersection from that bucket and add a new bucket containing the intersection (line 9). Finally, if the row of $\mathbb{A}$ is a super-set of some buckets, we add the part of the row that is not covered by the buckets as a new bucket (line 10).

The TRANSFORMSTRATEGY function in Algorithm 4 transforms $\mathbb{A}$ to $\mathbf{A}$, using the transformation matrix $\mathbb{T}$ to determine which buckets are used for each query. We first initialize $\mathbf{A}$ to be the zero matrix (line 15). For row i of $\mathbb{A}$ and row j of $\mathbb{T}$, we compute whether bucket j is needed to answer row i by checking if $\mathbb{T}[j] \subseteq \mathbb{A}[i]$ (line 17). If bucket j is needed, we set the corresponding entry of $\mathbf{A}$ to 1 (line 18).

Example 5.2. We consider $\mathbb{A}_2$ from example 3.1. The domain vector $\mathbb{x}$ consists of the leaves of the tree depicted in Figure 2. We get the following raw matrix form for $\mathbb{A}_2$.

$$\mathbb{A}_2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \mathbf{A}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We generate the full-rank form $\mathbf{A}_2$ above using $\mathbb{T}$:

$$\mathbb{T} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

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 $\mathbb{A}$.*

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 $\mathbf{WA}^+$).

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 <u>additional</u> cached strategy entries out of $C - \mathbb{A}$, to $\mathbb{A}$ to save more privacy budget.

Example 5.3. Consider the cache structure $C_{\mathbb{A}^*}$ in Figure 2 and a new workload $\mathbb{W}_1 = \{[0, 1), [0, 2), [2, 4)\}$ and so $\mathbb{A}_1 = \{\mathbb{A}^*_{[0,1)}$, $\mathbb{A}^*_{[1,2)}$, $\mathbb{A}^*_{[2,4)}\}$. The cache includes entries for $\mathbb{A}^*_{[0,1)}$, $\mathbb{A}^*_{[1,2)}$ at noise parameter b, as well as $\mathbb{A}^*_{[0,4)}$ at $4b$. The MMM module decides to reuse the first two cache entries, and pay for $\mathbb{A}^*_{[2,4)}$ at $5b$, resulting in the noise parameter vector $\mathbf{b}_1$, as depicted in Figure 5. The SE problem is deciding which cached responses (such as $\mathbb{A}^*_{[0,4)}$) can be added to the strategy to reduce it's cost.

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 $\mathbb{A}_e$. Prior work by Li et al. [18, Theorem 6] suggests that adding more queries to $\mathbf{A}$ always reduces the error of the matrix mechanism. However, their result hinges on the assumption that all strategy queries are answered using i.i.d draws from the <u>same</u> Laplace distribution [18]. Our cached strategy noisy responses can be drawn at different noise parameters in the past, and thus Li et al's result does not hold. In our case, the error term for the expanded strategy is given by: $\mathbf{WA}_e^+ \text{diag}(\mathbf{b}_e)$ (Proposition 4.2). In Figure 5, we present a counterexample for Li et al.'s result.

Example 5.4. Continuing Example 5.3, we expand $\mathbf{A}_1$ to $\mathbf{A}_{1e}$ by adding a row $\mathbb{A}^*_{[0,4)}$. In Figure 5, we compute the α^2-expected error using both $\mathbf{A}_1$ and $\mathbf{A}_{1e}$ and find that $\mathbf{A}_{1e}$ has a larger error term.

We can see that the strawman solution can lead to a strategy with an increased error term. Importantly, this figure shows that adding a strategy query results in changed coefficients in $\mathbf{WA}_e^+$, 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 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 3).

In an optimal solution to this problem, one would have to consider adding each combination of cache entries from $C - \mathbb{A}$. This

$$\mathbf{W}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \ \mathbf{A}_1 = I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \ \mathbf{b}_1 = \begin{bmatrix} b \\ b \\ 5b \end{bmatrix}$$

$$\mathbf{A}_{1e} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 1 & 1 & 1 \end{bmatrix}, \ \mathbf{b}_{1e} = \begin{bmatrix} b \\ b \\ 5b \\ \hline 4b \end{bmatrix}$$

$$\mathbf{W}_1 \mathbf{A}_1^+ \text{diag}(\mathbf{b}_1) = \begin{bmatrix} 1.00\,b & 0 & 0 \\ 1.00\,b & 1.00\,b & 0 \\ 0 & 0 & 5.00\,b \end{bmatrix}$$

$$\mathbf{W}_1 \mathbf{A}_{1e}^+ \text{diag}(\mathbf{b}_{1e}) = \begin{bmatrix} 0.750\,b & -0.250\,b & -1.25\,b & 1.00\,b \\ 0.500\,b & 0.500\,b & -2.50\,b & 2.00\,b \\ -0.250\,b & -0.250\,b & 3.75\,b & 1.00\,b \end{bmatrix}$$

$$\|\mathbf{W}_1 \mathbf{A}_1^+ \text{diag}(\mathbf{b}_1)\| = 28b^2, \ \|\mathbf{W}_1 \mathbf{A}_{1e}^+ \text{diag}(\mathbf{b}_{1e})\| = 29.1b^2$$

Figure 5: An input instant strategy $\mathbf{A}_1$ is expanded to a strategy $\mathbf{A}_{1e}$. Using their noise parameter vectors ($\mathbf{b}_1$, $\mathbf{b}_{1e}$), we can see that $\mathbf{A}_{1e}$ has an error term of larger magnitude.

induces an exponentially large search space of $O(2^{|C|})$ possible solutions for $\mathbb{A}_e$. Then for each candidate $\mathbb{A}_e$ we need to evaluate this error term and compare it to the error for the original strategy. We provide an example in Figure 5. Instead of navigating this large search space for $\mathbb{A}_e$, we propose a series of efficient heuristics to obtain a greedy solution to this problem, as presented in Algorithm 5. In designing our algorithm, we have three goals:

(1) *Search space:* Reduce the search space from $O(2^{|C|})$ to $O(|C|)$.
(2) *Efficiency:* Ensure that the additional strategy rows do not significantly increase the run-time of *CacheDP*.
(3) *Greediness:* Select strategy rows that are most likely to reduce the privacy budget from that for MMM ($\epsilon_\mathbf{P}$).

We achieve the first goal above by conducting a single lookup over cache entries in $C - \mathbb{A}$ (Line 3), which would only incur a worst-case complexity of $O(|C|)$. We limit the number of selected cached strategy rows to λ (Line 4), thereby achieving our second goal of efficiency. Our greediness heuristics to select a strategy query are based on the two aforementioned factors that impact the workload error term, namely, the <u>accuracy</u> of its cached noisy response, and how the noisy response is <u>related</u> to noisy responses to the original strategy.

First, we must ensure that the strategy queries selected from $C - \mathbb{A}$ are accurate enough. Before conducting our cache lookup, we sort our cache entries in increasing order of the noise parameter, therefore our algorithm greedily prefers more accurate cache entries. 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. Thus, out of cache entries in $C - \mathbb{A}$, we only consider cache entries whose noise parameter is lower than $b_\mathbf{P}$ (Line 4).

Second, our heirarchical global strategy $\mathbb{A}^*$ structures cache entries, and induces relations between the cached noisy responses.

Algorithm 5 Strategy Expander (SE) (Section 5.2)

1: **function** GENERATEEXPANDEDSTRATEGY($\mathbb{A}, C, b_\mathbf{P}$)
2:     $\mathbb{A}_e \leftarrow \mathbb{A}$
3:     **for** $(\mathbb{c}, b, \tilde{y}, t) \in (C - \mathbb{A})$ in an ascending order of b **do**
4:         **if** $|\mathbb{A}_e| \geq |\mathbb{A}| + \lambda$ or $b > b_\mathbf{P}$ **then**
5:             Break
6:         **if** $\mathbb{c}' \cdot \mathbb{c} \neq 0$ for some $\mathbb{c}' \in \mathbb{A}$ **then**
7:             $\mathbb{A}_e \leftarrow \mathbb{A}_e \cup \{\mathbb{c}\}$
8:     $(\mathbf{A}_e, \mathbb{T}_e) \leftarrow$ ST.TRANSFORMSTRATEGY($\mathbb{A}_e$)   ▷ Section 5.1
9:     **return** $\mathbb{A}_e, \mathbf{A}_e, \mathbb{T}_e$

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. [13]. 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. Thus, our second greedy heuristic, in line 6, ensures that each query $\mathbb{c}$ added to $\mathbb{A}_e$ is a parent or a child of an existing query $\mathbb{c}' \in \mathbb{A}$.

The SE algorithm generates an expanded strategy $\mathbb{A}_e$ and transforms it to its full-rank form $\mathbf{A}_e$ (Line 8). 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 (as illustrated in Example 5.4). Since Algorithm 1 chooses to run 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 evaluate the success of our heuristics both experimentally and theoretically in Appendix D.

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 $b_\mathbf{P}$.

We present the function SEARCHPROACTIVENODES for generating $\Delta\mathbb{P}$ in Algorithm 6. We formulate this algorithm in terms of the the k-ary tree representation of the global strategy $\mathbb{A}^*$, denoted by $\mathcal{T}$. (We assume this is a directed tree with directed edges from the root to leaves and all paths refer to paths from a node to its leaves.) For a node $v \in \mathcal{T}$, we define a binary function $\mathcal{M}_\mathbb{P}(n)$ to indicate if its corresponding query v.query is in $\mathbb{P}$.

Definition 5.1. We define the subtree norm of a node v as the maximum number of marked nodes across all paths p from node v-to-leaf in the subtree of v, i.e.,

$$\mathcal{S}_\mathbb{P}(v) = \max_{p \in \text{subtree}(v)} \sum_{v \in p} \mathcal{M}_\mathbb{P}(v) \tag{9}$$

The subtree norm of a node can be computed recursively as the sum of the mark function for that node and the maximum subtree norm of all of its children nodes, if any. The proactive module first recursively computes the subtree norm of each node before generating $\Delta\mathbb{P}$. This step requires a single top-down traversal of the strategy decomposition tree. Given that the children of each node have non-overlapping ranges, the subtree norm enables us to define the L_1 norm of $\mathbb{P}$:

Lemma 5.1. *The L_1 norm of the $\mathbb{P}$ matrix is equal to the subtree norm of the root of the tree with marked nodes corresponding to $\mathbb{P}$:*

$$\mathcal{S}_\mathbb{P}(\mathcal{T}.\text{root}) = \|\mathbb{P}\|_1 \tag{10}$$

Our proactive strategy generation function, GENERATEPROACTIVESTRATEGY, is presented in Algorithm 6. This function conducts a recursive top-down traversal of the strategy decomposition tree (line 9), and outputs a list of nodes to be fetched proactively into $\Delta\mathbb{P}$, such that the sum of the number of marked nodes and proactively fetched nodes for each path in the tree is at most $\|\mathbb{P}\|_1$.

Lemma 5.2. *The proactive strategy $\Delta\mathbb{P}$ generated by GENERATEPROACTIVESTRATEGY for an input $\mathbb{P}$ satisfies the condition:*

$$\forall \text{ paths } p \in \mathcal{T}, \sum_{v \in p} \mathcal{M}_{\mathbb{P} \cup \Delta\mathbb{P}}(v) \leq \mathcal{S}_\mathbb{P}(\mathcal{T}.\text{root}) = \|\mathbb{P}\|_1 \tag{11}$$

The second argument r in the function SEARCHPROACTIVENODES represents the number of *remaining* nodes that can be fetched proactively for the subtree originating at node v. Thus in the first call, we pass the root node for the first argument, and the second argument is initially set to $\|\mathbb{P}\|_1$. We decrement r whenever we encounter a marked node (line 2) or when we add a node to the proactive output list (line 5). In the latter case, we require that the node is not cached and that r is greater than the subtree norm of the node, $\mathcal{S}_\mathbb{P}(v)$ or s, as seen in line 4. This condition ensures that we can safely add node v to the proactive list, while achieving Equation (13). (We prove all PQ module lemmas and theorems in Appendix A.4.)

Theorem 5.2. *Given a paid strategy matrix $\mathbb{P}$ Algorithm 6 outputs $\Delta\mathbb{P}$ such that $\|\mathbb{P} \cup \Delta\mathbb{P}\|_1 = \|\mathbb{P}\|_1$.*

Algorithm 6 Proactive Querying (PQ) (Section 5.3)

1: **function** SEARCHPROACTIVENODES(node $v, r, C, \Delta\mathbb{P}$)
2:     **if** v.query $\in \mathbb{P}$ **then**
3:         $r \leftarrow r - 1$
4:     **else if** v.query $\notin C$ and $v.s < r$ **then**
5:         $\Delta\mathbb{P} \leftarrow \Delta\mathbb{P} \cup \{v.\text{query}\}$
6:         $r \leftarrow r - 1$
7:     **if** $r > 0$ and v has children **then**
8:         **for** child c of node v **do**
9:             SEARCHPROACTIVENODES(node $c, r, C, \Delta\mathbb{P}$)
10:    **return**

Algorithm 7 Relax Privacy (RP) (Section 6)

1: **function** ANSWERWORKLOAD($C, \mathbb{A}, \mathbf{W}, \varkappa$)
2:     $\eta_o \leftarrow \tilde{\mathbf{y}}_o - \mathbb{A}_o\varkappa$ ▷ Old noise vector for $\mathbb{A}_o$.
3:     $\eta \leftarrow$ LAPNOISEDOWN(η_o, b_o, b)   ▷ Koufogiannis et al. [17]
4:     $\tilde{\mathbf{y}} \leftarrow \mathbb{A}_o\varkappa + \eta$ ▷ New noisy responses to $\mathbb{A}_o$
5:     Update cache $C_{\mathbb{A}^*}$ with ($\mathbb{A}_o, b, \tilde{\mathbf{y}}, t$=current time)
6:     $\tilde{\mathbf{y}}' \leftarrow \tilde{\mathbf{y}}$ for $\mathbb{A} \subseteq \mathbb{A}_o$ ▷ New noisy responses to $\mathbb{A}$
7:     **return** $\mathbf{W}\mathbb{A}^+\tilde{\mathbf{y}}'$

8: **function** ESTIMATEPRIVACYBUDGET($C, \mathbb{A}, \mathbf{W}, \alpha, \beta$)
9:     $b \leftarrow$ MMM.ESTIMATEPRIVACYBUDGET($C = \emptyset, \mathbb{A}, \mathbf{W}, \alpha, \beta$)
10:    $C \leftarrow \{\cdots(\mathbb{A}_t, \tilde{\mathbf{y}}_t, b_t)\}$ ▷ 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:    $o = \arg\min\limits_{j} \; \epsilon_{RP,j} = \frac{\|\mathbb{A}_j\|_1}{b} - \frac{\|\mathbb{A}_j\|_1}{b_j}$ ▷ $\mathbb{A}_o$ has the lowest RP cost
15:    **return** b_o, $\epsilon_{RP,o}$ ▷ Cached noise parameter, RP cost for $\mathbb{A}_o$

Example 5.5. In Figure 2, 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. We annotate each node with the values of r and $v.s$ from line 4 in function SEARCHPROACTIVENODES. The tree nodes $\mathbb{A}^*_{[4,8)}$, $\mathbb{A}^*_{[0,2)}, \mathbb{A}^*_{[0,1)}, \mathbb{A}^*_{[1,2)}, \mathbb{A}^*_{[2,3)}$ and $\mathbb{A}^*_{[7,8)}$ satisfy the condition $r > v.s$. All nodes other than $\mathbb{A}^*_{[7,8)}$ are output into $\Delta\mathbb{P}_2$; the latter node is excluded since it is cached from $\mathbb{A}_1$ for $\mathbb{W}_1$. 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. However, $S_{\mathbb{P}_2}(\mathcal{T}) = S_{\mathbb{P}_2 \cup \Delta\mathbb{P}_2}(\mathcal{T}) = 2$.

5.3.1 Integration. The MMM ANSWERWORKLOAD function perturbs $\Delta\mathbb{P}$ with the same noise parameter as for $\mathbb{P}$, namely $b_\mathbf{P}$, to obtain the noisy responses $\tilde{\mathbf{y}}_{\Delta\mathbb{P}}$. It then updates the cache C with $\{(\mathbb{p}', b_\mathbf{P}, \tilde{y}, t)| \mathbb{p}' \in \Delta\mathbb{P}, \tilde{y} \in \tilde{\mathbf{y}}_{\Delta\mathbb{P}}\}$. We observe that we do not answer the analyst's workload query $\mathbf{W}$ using $\tilde{\mathbf{y}}_{\Delta\mathbb{P}}$. Importantly, this is the reason why we do not incorporate $\Delta\mathbb{P}$ in estimating $b_\mathbf{P}$ in our MMM cache-aware cost estimation function. The PQ module can also be used while the SE module is turned on. Algorithm 6 can also be applied to multi-attribute strategies, as we discuss in Section 7.

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 repeated workload can also be asked by a different analyst.) 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. [17] 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 $\epsilon_2 - \epsilon_1$ [17, 25]. However, their framework only operates with the simple Laplace mechanism. Thus, we achieve the aforementioned goal by closely integrating their framework [17] with the matrix mechanism and our DP cache. This design of the RP module meets a secondary goal, namely, our RP module handles not only repeated analyst-supplied *workloads*, but also different workloads that result in identical strategy matrices. We exploit the fact that other modules, such as the PQ module, also operate over the matrix mechanism and DP cache. Therefore, our RP module can be seen as generalizing these frameworks to operate over more workload sequences.

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 $\tilde{\mathbf{y}}$ 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 $\mathbf{W}, \mathbb{A}$. 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 $\mathbb{q} \in \mathbb{A}$ 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 2 have been asked in the past at α_1, and have been answered through MMM, as discussed in Example 5.5. 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 $\tilde{\mathbf{y}}_j$ 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).

Importantly, we only seek to obtain accuracy guarantees over $\mathbf{WA}^+$, and not over $\mathbf{WA}_o^+$, and thus we compute the target noise parameter b based on the input strategy matrix $\mathbb{A}$ (Line 9), rather than the optimal cached candidate $\mathbb{A}_o$. 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. The ESTIMATEPRIVACYBUDGET interface records the following parameters: the optimal cached or old strategy to relax ($\mathbb{A}_o$), its cached noisy response ($\tilde{y}_o$), the cached noise parameter (b_o) and the target noise parameter (b). We first compute the Laplace noise vector used in the past $\boldsymbol{\eta}_o$, by subtracting the ground truth for the cached old strategy $\mathbb{A}_o \rtimes$ 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) [17, 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 $\rtimes$ 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, $|\rtimes|$ 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 $\rtimes_{S_{\mathcal{A}}} = \bigotimes_{i=1}^{k \leq d} \rtimes_{\mathcal{A}_i}$. 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 $\rtimes_{S_{\mathcal{A}}}$.

Example 7.1. Consider three attributes with $dom(\mathcal{A}_1) = [0, 4)$, $dom(\mathcal{A}_2) = [0, 8)$, and $dom(\mathcal{A}_3) = [0, 2)$. The analyst supplied a workload sequence of RCQ over different sets of attributes: $\mathbb{W}_1$ over $S_1 = \{\mathcal{A}_1, \mathcal{A}_2\}$, $\mathbb{W}_2$ over $S_2 = \{\mathcal{A}_2, \mathcal{A}_3\}$ and then $\mathbb{W}_3$ over $S_3 = S_1 = \{\mathcal{A}_1, \mathcal{A}_2\}$. Then $\mathbb{W}_1$ and $\mathbb{W}_3$ are answered using the same domain vector $\rtimes_{S_1} = \rtimes_{\mathcal{A}_1} \otimes \rtimes_{\mathcal{A}_2}$ and a cache indexed over it, C_{S_1}. However, $\mathbb{W}_2$ is answered using a different domain vector $\rtimes_{S_2} = \rtimes_{\mathcal{A}_2} \otimes \rtimes_{\mathcal{A}_3}$ and the corresponding cache C_{S_2}.

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.

For a given workload $\mathbb{W}$ that spans a set of attributes $S_{\mathcal{A}}$, we can use the single-attribute strategy tree $\mathcal{T}_i$ for each attribute $\mathcal{A}_i \in S_{\mathcal{A}}$ to construct a multi-attribute strategy tree $\mathcal{T}$ for $\mathbb{W}$, as follows. Intuitively, $\rtimes$ consists of marginals that are constructed by taking each unit value in $dom(\mathcal{A}_i)$, and forming a cross product with each unit value in the $dom(\mathcal{A}_{i+1})$, and so on. Unit values in $dom(\mathcal{A}_i)$ are represented by leaf nodes on its strategy tree $\mathcal{T}_i$, as illustrated in Figure 2. Thus, we form a two-attribute strategy tree $\mathcal{T}$ over $\mathcal{A}_1$ and $\mathcal{A}_2$, by attaching the tree $\mathcal{T}_2$ to each leaf node for the tree $\mathcal{T}_1$. (We can then simply extend this definition to $k = \|S_{\mathcal{A}}\|$ attributes in $\mathbb{W}$.) Nodes on $\mathcal{T}$ either represent marginals over these attributes or sums of marginals. In particular, the leaf nodes on $\mathcal{T}$ represent the unit value strategy marginals over $\mathcal{A}_1$ and $\mathcal{A}_2$.

We can thus decompose each workload marginal $\mathbb{w}$ in terms of nodes on this tree $\mathcal{T}$. In the first step, we decompose each single-attribute marginal $\mathbb{w}_i$ following our single-attribute DECOMPOSEWORKLOAD function (Algorithm 4) to get a set of single-attribute strategy nodes $\mathbb{o}_i$. The second step is run for all but the last (kth) attribute: in this step, we further decompose $\mathbb{o}_i$ in terms of the leaf nodes on tree $\mathcal{T}_i$, to obtain $\mathbb{o}_{\text{leaf},j}$. In the final step, we compute the cross-product of the single-attribute strategy nodes $\mathbb{o}_{\text{leaf},j}$, over all j, and a final cross product with the decomposition of the kth attribute, $\mathbb{o}_k$, to form the strategy marginal $\mathbb{o}$ for the workload marginal $\mathbb{w}$, as follows: $\mathbb{o} = \bigotimes_{i=1}^{k-1 \leq d} \mathbb{o}_{\text{leaf},j} \otimes \mathbb{o}_k$.

To determine the order in which to decompose the attributes, we first compute the minimum granularity needed for each attribute to be able to answer the workload. We then sort the attributes in acceding order of granularity. The intuition is that this should result in the least amount of noise compounding for the lowest attribute on the tree. Each decomposition is greedy in that we only go down to the minimum granularity for the given workload and not to the leaf nodes. This further reduces the noise compounding of our approach.

We design the above strategy transformer such that our PQ module can easily be applied to the structure. The only difference is that the GENERATEPROACTIVESTRATEGY algorithm is run over the strategy tree for the combination of attributes in the workload ($S_{\mathcal{A}}$) rather than the global tree structure. The PQ module can thus add queries for any subset of attributes in $S_{\mathcal{A}}$, prioritizing nodes higher up in the tree, which correspond to fewer number of attributes. In practice, this is preferable as it focuses on expanding the areas an analyst has already shown interest in rather than completely unrelated attributes.

Example 7.2. Consider the first workload in the sequence in Example 7.1. It consists of one marginal tuple over attributes $\mathcal{A}_1$ and $\mathcal{A}_2$: ($\mathbb{W} = [0, 3)_{\mathcal{A}_1}, [1, 6)_{\mathcal{A}_2}$). We denote the global strategy matrix for attribute i as $\mathbb{A}i^*$. We decompose the RCQ for each attribute in

Dataset	ADULT [15]	TAXI [1]	PLANES [2, 8]
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.

the marginal, using our single attribute strategy generator. Thus: $\mathbb{A}_{\mathcal{A}1} = \{\mathbb{A}1^*_{[0,2)}, \mathbb{A}1^*_{[2,3)}\}$ and $\mathbb{A}_{\mathcal{A}2} = \{\mathbb{A}2^*_{[1,2)}, \mathbb{A}2^*_{[2,4)}, \mathbb{A}2^*_{[4,6)}\}$. We then construct the two-attribute strategy marginals as the cross product of these RCQs: $\mathbb{A} = \mathbb{A}_{\mathcal{A}1} \bigotimes \mathbb{A}_{\mathcal{A}2} = \{(\mathbb{A}1^*_{[0,2)}, \mathbb{A}2^*_{[1,2)}),$ $(\mathbb{A}1^*_{[0,2)}, \mathbb{A}2^*_{[2,4)}), (\mathbb{A}1^*_{[0,2)}, \mathbb{A}2^*_{[4,6)}), (\mathbb{A}1^*_{[2,3)}, \mathbb{A}2^*_{[1,2)}), (\mathbb{A}1^*_{[2,3)}, \mathbb{A}2^*_{[2,4)}),$ $(\mathbb{A}1^*_{[0,2)}, \mathbb{A}2^*_{[4,6)})\}$

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 [11]: *APEx* is a state-of-the-art accuracy-aware interactive DP query engine. *APEx* consumes accuracy requirements in the form of an (α, β) bound (see Definition 2.2). *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 [25]: *Pioneer* is a DP query engine that incorporates a cache of previous noisy responses to save the privacy budget on future queries. *Pioneer* expects the accuracy over its responses in the form of a target variance. 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 Interactive Exploration Tasks. We use three datasets: the 1994 US Census data in the ADULT dataset [15] (48842 rows $\times$14 attributes), a log of NYC yellow taxi trip records from 2015 in the TAXI dataset [1] (1028527×19), and US domestic flight records in the PLANES dataset [2, 8] ($500,000 \times 12$). The TAXI dataset is used to model a single strategy tree over a pair of correlated (*Lat, Long*) attributes. Each node on the tree (or a range query) represents a rectangle of area on a map, and each level on the tree splits each node's rectangle into its quarters.

BFS and DFS tasks: A common data exploration task involves traversing a decomposition tree over the domain [32], by progressively asking more fine-grained queries over a subset of the domain. In each iteration, the analyst decomposes each query in the past workload whose noisy response satisfies a certain criteria, into k

new children queries on the attribute decomposition tree. In the BFS task, the analyst explores only past queries with a sufficiently *high* noisy count. The BFS task thus returns the smallest subsets of a domain that are sufficiently populated.

Whereas in a DFS task, the analyst focuses on past queries with a sufficiently *low non-zero* count (i.e. underrepresented subgroups). A DFS task terminates if a query's noisy count is non-zero and falls within the low DFS threshold range. In the DFS task, when the analyst reaches a leaf node without finding a node with a sufficiently low count, they backtrack a random number of steps up the tree, and resume the search with the second smallest node at that level. We group the range queries that satisfy the BFS or DFS criteria, into a single workload per level of the tree.

Randomized range queries (RRQ) over synthetic data: We replicate the evaluation of *Pioneer* [25] through $50,000$ randomized range queries over a synthetic dataset. We fix a domain size of $m = 1,000$ such that each range query is contained in $(0, m)$. Each query is in the form of $(s, s+\ell)$ where s and ℓ are both selected from a normal distribution with the following mean (μ) and standard deviation (σ): $\mu_s = 500$, $\sigma_s = 10$, $\mu_\ell = 320$, and $\sigma_\ell = 10$. The accuracy requirement is supplied as an expected square error, which is also selected from a normal distribution with $\mu = 250000, \sigma = 25000$.

IDEBench for Multi-Attribute queries: Eichman et al. develop a benchmarking tool to evaluate interactive data exploration systems [8]. We use this tool to construct a sequence of exploration workloads for a multi-attribute case study over the PLANES dataset. Specifically, we extract the SQL workloads for their $1 : N$ case where each query triggers an additional N dependent queries [8]. We simplify these queries for easy integration with our prototype.

8.1.3 Client Modelling. We model multiple data analysts querying the aforementioned systems, as 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.

8.1.4 Experiment setup. We run the BFS and DFS tasks with $c = 25$ clients and the IDEBench task with $c = 10$ clients. (We only run the RRQ task with a single client, to precisely replicate the Pioneer paper. Since *Pioneer* can only work with a single attribute, we do not run it for tasks over the TAXI or PLANES datasets.) We detail the accuracy requirements and task parameters in the full paper. The BFS and DFS tasks are conducted over the *Age* and *Country* attributes of the ADULT dataset respectively. Both tasks are also run over the *Latitude (Lat)* and *Longitude (Long)* attributes of the TAXI dataset. Each client randomly draws the minimum threshold for their BFS and the maximum threshold for their DFS. The BFS, DFS and IDEBench tasks consume (α, β) accuracy requirements, with fixed $\beta = 0.05$ across all clients. Each client randomly selects $\alpha = \alpha_s |D|$ for $\alpha_s \in [0.01, 0.16]$, with step size 0.05.

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

(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)
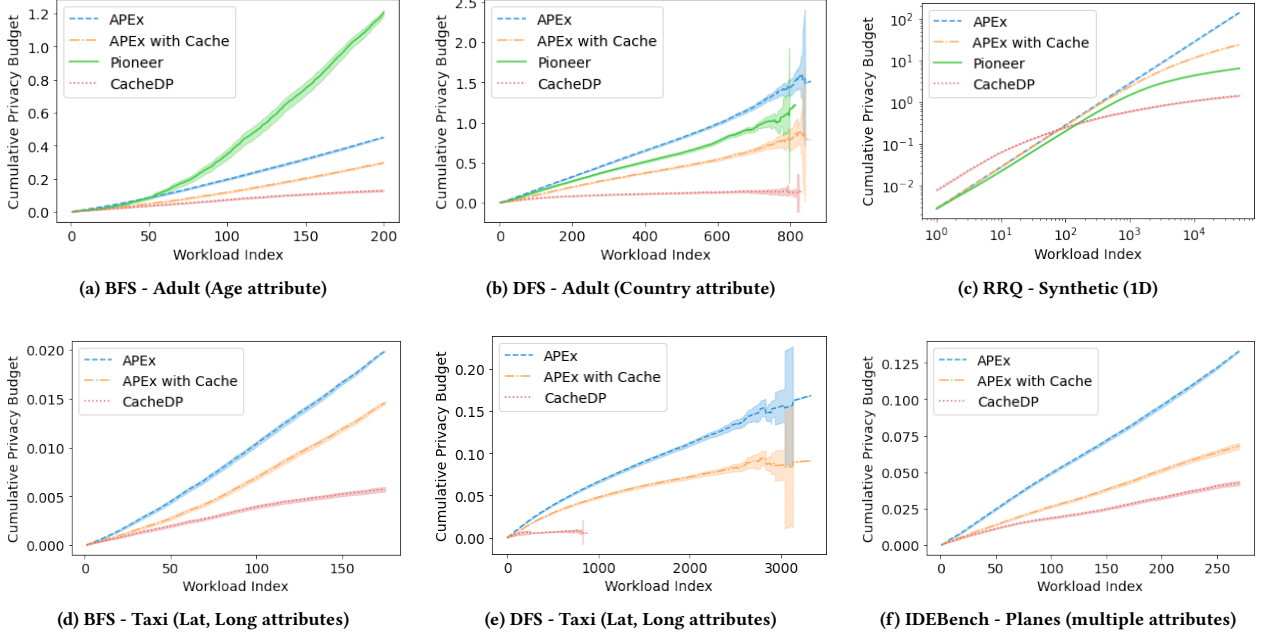
(f) IDEBench - Planes (multiple attributes)

Figure 6: Average cumulative privacy budget comparison between CacheDP and baselines (APEx, APEx with cache, Pioneer).

plot the mean and 95% confidence intervals in Figure 6. 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 6a, 6b). 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 6c). Our Pioneer implementation replicates a similar privacy budget trendline to the original paper [25, 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 6c), *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 6d, 6e) and multiple attributes (Figure 6f), *CacheDP* outperforms the best baseline (*APEx with cache*), by at least a factor of 1.5 for Figure 6f.

For both DFS tasks (Figures 6b, 6e), 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 8.1.4), and thus, we find that in Figure 6e, *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.

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
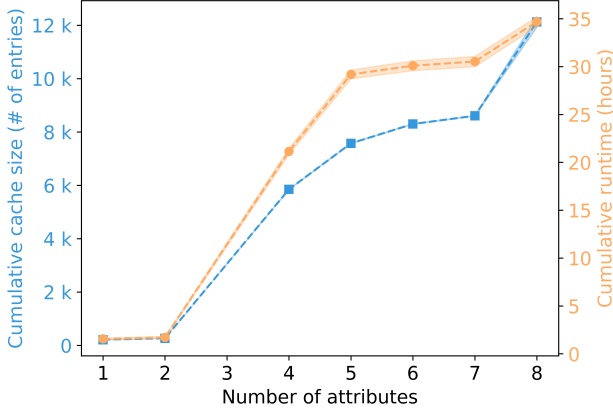
Figure 7: Average cumulative cache size and runtime of *CacheDP*, versus the number of attributes, for the IDEBench task.

(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. With an increase in the number of attributes, as observed in the IDEBench task, we find that the cumulative cache size and runtime of *CacheDP* scales linearly. Specifically, IDEBench workloads require computations over a larger data vector that spans many attributes. Yet, non-optimized *CacheDP* only takes around 6 minutes per workload for the IDEBench task, and performs slightly better than APEx with cache.

We illustrate how the cumulative cache size and runtime of *CacheDP* scales with the number of attributes for the IDEBench task in Figure 7. We note that the x-axis corresponds to a total of 27 IDEBench workloads. We consider multiple clients in Table 3, whereas we only consider one client in Figure 7.[1] We also note that increasing the number of records in the database only impacts the size of the data vector in the pre-processing stage, and does not impact the size of the cache.

We can see that both observed variables scale approximately linearly with the number of attributes. The cache size can be limited by identifying inaccurate cache entries that can be replaced, while inserting new noisy responses, in each module. The MC simulation forms the bottleneck for the runtime, when the number of attributes remains small, whereas the matrix multiplication becomes the main bottleneck as the number of attributes increases. The MC simulation can be avoided entirely, by expressing the desired accuracy

---

[1]Since this single client will not experience fully cached, accurate workload responses, we find an order of magnitude worse runtime overheads than simply by multiplying the average in Table 3 by the number of IDEBench workloads. Thus this plot shows worst-case runtimes for the IDEBench task.

through expected variances instead of the (α, β) requirements. Our implementation can be optimized with fast matrix multiplication algorithms to achieve smaller runtimes.

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 8. The standard configuration 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.
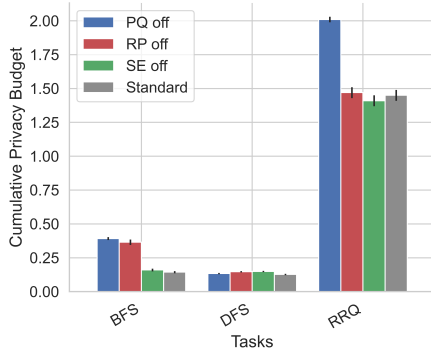
Fourth, turning the SE module off only contributes to minor differences in the cumulative privacy budget (B_c). However, the reader may expect that turning the SE module off would lead to a higher B_c, since based on the frequency analysis, the SE module is most frequently chosen to answer non-zero workloads for the BFS and DFS tasks. We reconcile this discrepancy as follows. When the SE module is chosen, constrained inference reduces the privacy

System	BFS - Adult (Age)		DFS - Adult (Country)		RRQ - Synthetic (1D)		BFS - Taxi (Lat,Long)		DFS - Taxi (Lat,Long)		IDEBench	
	Cache Entries	Runtime (s)	Cache Entries	Runtime (s)	Cache Entries	Runtime (s)	Cache Entries	Runtime (s)	Cache Entries	Runtime (s)	Cache Entries	Runtime (s)
APEx	-	3.7±0.2	-	2.71±0.02	-	0.05±0	-	111±8	-	25±1	-	456±70
APEx Cache	123±1	3.7±0.2	81±0	2.71±0.02	3118±4	0.05±0	668±14	111±8	2515±0	25±1	6540±0	456±70
Pioneer	117±1	4.8±0.2ms	80±0	1.6±0.2ms	3118±4	0.01±0	-	-	-	-	-	-
CacheDP	145±3	7.5±0.4	81±0	2.8±0.2	1998±0	0.05±0	3669±0	23±1	3669±0	15.3±0.5	23666±1000	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 8: Ablation study over PQ, RP, SE modules of *CacheDP*

cost for paid strategy queries. As a result, the cache entries for these queries remain at a lower accuracy than if MMM or RP had been used to answer them, and they may not be reusable for later workloads. Thus *CacheDP* may need to obtain noisy responses for these queries, later on, at a higher accuracy. The SE module thus provides savings on earlier workloads 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 [26, 32] and in synthetic data generators [12, 21, 27] 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 [28]. 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 [11, 23], releasing data [10, 24], or programming frameworks [29, 31] 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 [11].

Peng et al.'s Pioneer [25] is the most relevant work that uses historical query answers to obtain accurate responses to upcoming single range queries over the same or related predicates. 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 extension of the ST module (Section 7).

Our key modules are built on top of prior work (e.g., Li et al.'s Matrix Mechanism [18], Koufogiannis et al.'s Relax Privacy Mechanism [17]), such that existing interactive DP systems that make use of these mechanisms (e.g. PrivateSQL [16], APEx [11]) do not have to make significant changes; these systems can include a relatively light-weight cache structure and cache-aware version of the corresponding 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), and our extension of our one-dimensional cache to high-dimensional cache (Section 7).

10 FUTURE WORK:

CacheDP can be extended to answer top-k or iceberg counting queries studied in APEx, as well as simple aggregates such as means, by integrating the query processing engine of APEx to transform these queries to raw counting queries. Beyond counting queries, providing differential privacy over complex SQL queries, such as joins and group by operators, is a challenging problem, as studied in prior work [5, 16, 30]. The global sensitivity of SQL queries involving joins is unbounded. To tackle this challenge, the existing well-performed DP mechanisms [5, 16, 30] for these queries require a data-dependent transformation (e.g., truncation of the data) in order to bound the sensitivity of the query. Thus, the accuracy of these mechanisms depend on the data, and searching the minimum privacy budget to achieve the desired accuracy bound is non-trivial, which is an important research direction.

11 CONCLUSION

We build a usable interactive DP query engine, *CacheDP*, that uses a differentially private structured 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 knowledge of DP. 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).

REFERENCES

[1] [n.d.]. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.
[2] [n.d.]. https://transtats.bts.gov.
[3] 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
[4] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In Proceedings of the 31st International Conference on Neural Information Processing Systems (Long Beach, California, USA) (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 3574–3583.
[5] Wei Dong and Ke Yi. 2021. Residual Sensitivity for Differentially Private Multi-Way Joins. In Proceedings of the 2021 International Conference on Management of Data (Virtual Event, China) (SIGMOD '21). Association for Computing Machinery, New York, NY, USA, 432–444. https://doi.org/10.1145/3448016.3452813
[6] 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.
[7] 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
[8] 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.
[9] Marco Gaboardi, Michael Hay, and Salil Vadhan. 2019. A Programming Framework for OpenDP. https://projects.iq.harvard.edu/opendp
[10] Marco Gaboardi, James Honaker, Gary King, Jack Murtagh, Kobbi Nissim, Jonathan Ullman, and Salil Vadhan. 2016. Psi ({\Psi}): a private data sharing interface. arXiv preprint arXiv:1609.04340 (2016).
[11] 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) (SIGMOD '19). Association for Computing Machinery, New York, NY, USA, 177–194. https://doi.org/10.1145/3299869.3300092
[12] Chang Ge, Shubhankar Mohapatra, Xi He, and Ihab F. Ilyas. 2021. Kamino: Constraint-Aware Differentially Private Data Synthesis. VLDB (2021).
[13] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the Accuracy of Differentially Private Histograms through Consistency. Proc. VLDB Endow. 3, 1–2 (Sept. 2010), 1021–1032. https://doi.org/10.14778/1920841.1920970
[14] Noah Johnson, Joseph P. Near, and Dawn Song. 2018. Towards Practical Differential Privacy for SQL Queries. Proc. VLDB Endow. 11, 5 (Jan. 2018), 526–539. https://doi.org/10.1145/3177732.3177733
[15] 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.
[16] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: A Differentially Private SQL Query Engine. Proc. VLDB Endow. 12, 11 (July 2019), 1371–1384. https://doi.org/10.14778/3342263.3342274
[17] 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
[18] 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
[19] 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
[20] 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://dettanym.github.io/files/CacheDP-tech-report.pdf
[21] Ryan McKenna, Daniel Sheldon, and Gerome Miklau. [n.d.]. Graphical-model based estimation and inference for differential privacy. ([n. d.]). arXiv:1901.09136

http://arxiv.org/abs/1901.09136
[22] Frank McSherry. 2010. Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis. Commun. ACM 53, 9 (sep 2010), 89–97. https://doi.org/10.1145/1810891.1810916
[23] 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 (Scottsdale, Arizona, USA) (SIGMOD '12). Association for Computing Machinery, New York, NY, USA, 349–360. https://doi.org/10.1145/2213836.2213876
[24] 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
[25] 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
[26] 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
[27] 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).
[28] Yuchao Tao, Ryan McKenna, Michael Hay, Ashwin Machanavajjhala, and Gerome Miklau. 2021. Benchmarking Differentially Private Synthetic Data Generation Algorithms. TPDP (2021).
[29] Elisabet Lobo Vesga, Alejandro Russo, and Marco Gaboardi. 2019. A Programming Framework for Differential Privacy with Accuracy Concentration Bounds. CoRR (2019). arXiv:1909.07918
[30] 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
[31] 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
[32] 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 (San Francisco, California, USA) (SIGMOD '16). ACM, 155–170. https://doi.org/10.1145/2882903.2882928

A   PROOFS

A.1   End-to-end Privacy Proof

A.1.1   Proof of Theorem 3.1. Recall the theorem states that *CacheDP*, as defined in Algorithm 1, satisfies $\mathcal{B}$-DP.

PROOF. (sketch) We begin by addressing the cost estimation phase (lines 4-10). The cost estimation phases are independent of the data. In addition, each DP mechanism (MMM or RP) with its corresponding chosen strategy ensures ϵ_i-DP (Proposition 4.1, [17, Theorem 1A]). At line 11, we check if running the chosen DP mechanism (MMM or RP with the corresponding chosen strategy) will exceed the total privacy budget $\mathcal{B}$ by sequential composition [7]. We only run the DP mechanism if the total budget is sufficient. This ensures the overall Algorithm 1 satisfies $\mathcal{B}$-DP.

□

A.2   MMM Module Proofs

A.2.1   Proof of Proposition 4.1. Recall the proposition states that the ANSWERWORKLOAD interface of MMM (Algorithm 2) satisfies ϵ-DP, where ϵ is the output of this interface.

PROOF. (sketch) When the cache is empty, the privacy of MMM (with optional SE) follows from the matrix mechanisms privacy guarantee in Proposition 2.1. When there are entries in the cache, we split the strategy into the free and paid matrix. The paid matrix

is private by the same reasoning as above. The privacy of the free matrix responses and the final concatenation of free and paid responses follow by the post processing lemma of DP [7, Proposition 2.1]. □

A.2.2 Proof of Proposition 4.2. Given an instant strategy $\mathbf{A} = (\mathbf{F}||\mathbf{P})$ with a vector of k noise parameters $\mathbf{b} = \mathbf{b_F}||\mathbf{b_P}$, the error to a workload $\mathbf{W}$ using the AnswerWorkload interface of MMM (Algorithm 2) is $\|\mathbf{WA^+}Lap(\mathbf{b})\|$, 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 $\|\mathbf{WA^+}diag(\mathbf{b})\|_F^2$ where $diag(\mathbf{b})$ is a diagonal matrix with $diag(\mathbf{b})[i,i] = \mathbf{b}[i]$.

PROOF. The error to the MMM is $\|\mathbf{WA^+}(\mathbf{Ax} + Lap(\mathbf{b})) - \mathbf{Wx}\| = \|\mathbf{WA^+}Lap(\mathbf{b})\|$. The expected total square error $\mathbb{E}[\|\mathbf{WA^+}Lap(\mathbf{b})\|_2^2]$ can be expanded to $\sum_{i=1}^l \mathbb{E}[\sum_{j=1}^k (\mathbf{WA^+}[i,j]Lap(\mathbf{b}[j]))^2]$. As the k noise variables are independent and has a zero mean, then we have the error expression equals to

$$\sum_{i=1}^l \sum_{j=1}^k (\mathbf{WA^+}[i,j])^2 \mathbb{E}[(Lap(\mathbf{b}[j]))^2] = \sum_{i=1}^l \sum_{j=1}^k (\mathbf{WA^+}[i,j])^2 \mathbf{b}[j]^2$$

which is equivalent to $\|\mathbf{WA^+}diag(\mathbf{b})\|_F^2$. □

A.2.3 Proof of 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 $\mathbf{F} = \emptyset$.

PROOF. (sketch) Let b^* be the noise parameter for $\mathbf{A}$ in the matrix mechanism without cache to achieve the desired accuracy requirement. We can show that b^* is a valid solution to the simplified CE problem: setting $\mathbf{b_F} = [c.b \in C \mid c.a \in C \cap A, c.b \leq b^*]$ and $\mathbf{b_P} = [b^* \mid a \in \mathbf{P}]$ satisfies the accuracy requirement. As $\|\mathbf{P}\|_1 \leq \|\mathbf{A}\|_1$, the privacy cost $\epsilon = \frac{\|P\|_1}{b^*}$ for $b_{\mathbf{P}} = b^*$ is smaller than $\epsilon_{\mathbf{F}=\emptyset} = \frac{\|\mathbf{A}\|_1}{b^*}$. The optimal solution to the simplified CE problem has a smaller or the same privacy cost than a valid solution $b_{\mathbf{P}} = b^*$. □

A.3 Full-rank Transformer (FRT) Proof

Recall Theorem 5.1 states that 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 begin by proving the following lemma.

Lemma A.1. *Given a global strategy $\mathbb{A}^*$ in a k-ary tree structure, and an instant strategy $\mathbb{A} \subseteq \mathbb{A}^*$, running GETTRANSFORMATIONMATRIX($\mathbb{A}$) in Algorithm 4 increases the number of non-empty buckets in $\mathbb{T}$ by at most 1, for each row $\mathbb{A}[i] \in \mathbb{A}$.*

PROOF. If the condition in line 4 is met the result follows trivially. We consider the remaining case where $\mathbb{A}[i]$ intersects with at least one bucket. We recall that all entries in $\mathbb{A}$ represent nodes in a k-ary tree. Since adding and subtracting nodes on a k-ary tree always results in a combination of one or more nodes on a k-ary tree, we can conclude that at the end of each loop, all of the buckets are disjoint. For each new row $\mathbb{A}[i]$, if this row intersects with the buckets in $\mathbb{T}$, then $\mathbb{A}[i]$ is either (i) a descendant node of one bucket in $\mathbb{T}$, or (ii) an ancestor node of one or more buckets in $\mathbb{T}$. For the first case, assume $\mathbb{A}[i]$ is a descendant node of $\mathbb{t} \in \mathbb{T}$. It cannot be

the descendant of other buckets, as the other buckets are disjoint with $\mathbb{t}$. Then $\mathbb{t}$ will be replaced by $\mathbb{t} \cap \mathbb{A}[i]$ and $\mathbb{t} - \mathbb{A}[i]$ (line 9), and hence the size of $\mathbb{T}$ increases by 1. For the second case, assume $\mathbb{A}[i]$ is the ancestor node of multiple buckets in $\mathbb{T}$. We denote these buckets as $\{\mathbb{T}[j_1], \ldots, \mathbb{T}[j_k]\}$, then all these buckets will remain the same (line 9 just adds $\mathbb{t}$), and at most one additional bucket $\mathbb{A}[i] - \sum_{\mathbb{t} \in \mathbb{T}' \wedge \mathbb{t} \cdot \mathbb{A}[i] \neq 0} \mathbb{t}$ is added in line 10. □

We now prove the main result (Theorem 5.1) that TRANSFORM-STRATEGY ensures full rank matrices.

PROOF. We begin by discussing how $\mathbf{A}$ represents the queries in $\mathbb{A}$. First, we note that $\mathbf{A}$ and $\mathbb{A}$ have the same number of rows. The only difference is that $\mathbf{A}$ represents the queries on the data vector $\mathbf{x}$ where as $\mathbb{A}$ uses $\mathbb{x}$. By construction, we have $\mathbb{A}\mathbb{x} = \mathbf{A}\mathbb{T}\mathbb{x} = \mathbf{Ax}$.

Since $\mathbf{A}$ is a representation of $\mathbb{A}$ using the only the buckets generated in GETTRANSFORMATIONMATRIX, the number of columns in $\mathbf{A}$ is the same as the number of buckets, $|\mathbb{T}|$. To show that $\mathbf{A}$ is always full rank, we will first show that $|row(\mathbf{A})| \geq |col(\mathbf{A})|$, where $|row(\mathbf{A})|$ and $|col(\mathbf{A})|$ represent the number of rows and the number of columns in $\mathbf{A}$. Or equivalently, $|row(\mathbf{A})| \geq |\mathbb{T}|$. This follows by inductively applying Lemma A.1.

Next we show that $rank(\mathbf{A}) = |col(\mathbf{A})|$. We once again show this by induction on the number of rows in $\mathbb{A}$. In the base case ($|\mathbb{A}| = 1$) applying TRANSFORMSTRATEGY, we get $\mathbf{A} = [1]$ and thus $rank(\mathbf{A}) = 1$. Now we assume that $rank(\mathbf{A}) = |col(\mathbf{A})|$ for some $\mathbf{A}$ obtained from $\mathbb{A}$. We consider adding a new row r_{new} to $\mathbb{A}$ and define $\bar{\mathbb{A}} = \mathbb{A} \cup r_{new}$. Let $\bar{\mathbf{A}}$ represent the result of applying TRANSFORMSTRATEGY to $\bar{\mathbb{A}}$.

When adding this row there are two possible cases: First, consider the case where the newly added row, r_{new}, can be represented as a linear combination of $\mathbb{T}$. That is the buckets created in GET-TRANSFORMATIONMATRIX are the same for $\mathbb{A}$ and $\bar{\mathbb{A}}$ In this case, $\bar{\mathbf{A}} = \mathbf{A} \cup r'_{new}$. Thus $|col(\bar{\mathbf{A}})| = rank(\bar{\mathbf{A}})$ since the number of linearly independent columns (or buckets) did not increase or decrease by adding a row. On the other hand, if the newly added row, r_{new}, cannot be represented as a linear combination of $\mathbb{T}$, then r_{new} must be linearly independent of $\mathbb{A}$. Thus, $\bar{\mathbb{A}}$ has one additional linearly independent row and thus $rank(\bar{\mathbf{A}}) = rank(\mathbf{A}) + 1$. Furthermore by Lemma A.1, we know that adding a row can add at most one new bucket. Since we assume r_{new} cannot be represented as a linear combination of $\mathbb{T}$, this means $|col(\bar{A})| = |col(A)| + 1$. Thus we have that $|col(\bar{A})| = rank(\bar{A})$. Combining the fact that $|row(\mathbf{A})| \geq |col(\mathbf{A})|$ and $rank(\mathbf{A}) = |col(\mathbf{A})|$, it follows that $\mathbf{A}$ is full rank. □

A.4 PQ Module Proofs

A.4.1 Proof of Lemma 5.1. Recall the lemma states that the L_1 norm of the $\mathbb{P}$ matrix is equal to the subtree norm of the root of the tree with marked nodes corresponding to $\mathbb{P}$:

$$\mathcal{S}_{\mathbb{P}}(\mathcal{T}.\text{root}) = \|\mathbb{P}\|_1 \tag{12}$$

PROOF. Given that we form $\mathbb{P}$ as shown in Example 5.2, $\|\mathbb{P}\|_1$ simply represents the maximum number of overlapping RCQs in $\mathbb{P}$. Overlapping RCQs on the tree $\mathcal{T}$ must occur on the same path, that is, they form an ancestor-descendant relationship, since the children of each node n have non-overlapping ranges. Thus, the

maximum number of overlapping RCQs across all tree paths, is equal to the $\|\mathbb{P}\|_1$. □

A.4.2 Proof of Lemma 5.2.
The lemma states that the proactive strategy $\Delta\mathbb{P}$ generated by GENERATEPROACTIVESTRATEGY for an input $\mathbb{P}$ satisfies the condition:

$$\forall \text{ paths } p \in \mathcal{T}, \sum_{v \in p} \mathcal{M}_{\mathbb{P} \cup \Delta\mathbb{P}}(v) \leq \mathcal{S}_{\mathbb{P}}(\mathcal{T}.\text{root}) = \|\mathbb{P}\|_1 \quad (13)$$

PROOF. This inequality holds for $\Delta\mathbb{P} = \emptyset$ by applying Lemma 5.1 and Definition 5.1. GENERATEPROACTIVESTRATEGY only adds a node to $\Delta\mathbb{P}$ if it satisfies the following condition on line 4, the remaining path has a length less than r. At the root, this condition is set to be less than $\|\mathbb{P}\|_1$. □

A.4.3 Proof of Theorem 5.2.
The theorem states that, given a paid strategy matrix $\mathbb{P}$, Algorithm 6 outputs $\Delta\mathbb{P}$ such that $\|\mathbb{P} \cup \Delta\mathbb{P}\|_1 = \|\mathbb{P}\|_1$.

PROOF. Applying Lemma 5.1 and Definition 5.1 for the matrix $\mathbb{P} \cup \Delta\mathbb{P}$:

$$\|\mathbb{P} \cup \Delta\mathbb{P}\|_1 = \mathcal{S}_{\mathbb{P} \cup \Delta\mathbb{P}}(\mathcal{T}.\text{root}) = \max_{p \in \text{subtree}(v)} \sum_{v \in p} \mathcal{M}_{\mathbb{P} \cup \Delta\mathbb{P}}(v) \quad (14)$$

Rephrasing Lemma 5.2:

$$\max_{p \in \text{subtree}(v)} \sum_{v \in p} \mathcal{M}_{\mathbb{P} \cup \Delta\mathbb{P}}(v) = \|\mathbb{P}\|_1 \quad (15)$$

Using equations 14 and 15, we get:

$$\|\mathbb{P} \cup \Delta\mathbb{P}\|_1 = \|\mathbb{P}\|_1 \quad (16)$$

□

B MMM CACHE-AWARE TIGHT BOUND

Given the cached noise parameters, we propose a theoretical upper bound b_T for the candidate b_{P0} required to satisfy the (α, β)-accuracy guarantee.

$$b_T \leq \frac{\sqrt{\alpha^2\beta/2 - \|\mathbf{WA}^+ Diag(\vec{b_{\mathbf{F}}})\|_F^2}}{\|WA^+ Diag(I_{|\mathbf{P}|})\|_F} \quad (17)$$

Here, $\vec{b_{\mathbf{F}}} = [b_1, \ldots, b_{|\mathbf{F}|}]$. In doing so, we generalize Ge et al.'s tight bound b_T for $C = \phi$ (equation 4) to consider cached noise parameters.

C IMPLEMENTATION DETAILS

We implement an initial prototype of *CacheDP* in Python. We use the source code provided by Ge et al. to evaluate APEx[2]. Since the authors of Pioneer did not publish their code, we implement it from scratch in python following the paper. We implement a simple composite plan using all hyperparamters as described in the paper. We show in Section 8.2.1 that our implementation reproduces similar performance to the results given in the original paper [25][Section 7.2.1]. We make our full evaluation including our implementation of related work publicly available[3].

We note that pioneer uses variance based accuracy where as APEx uses (α, β) accuracy requirements. This is not a problem

for *CacheDP* as we accept both types of accuracy requirement. To run Pioneer on workloads with an (α, β) requirement, we use an MC simulation to search for the variance that satisfies the accuracy requirement. To run APEx on workloads with a variance based accuracy requirement we utilize a tail bound on the Laplace distribution (since we find empirically that apex uses the Laplace Mechanism for all single range queries) to get the alpha beta.

D EVALUATION OF THE SE HEURISTICS

In this section, we reason about the effectiveness of the strategy expander heuristics, both experimentally and theoretically.

Experiments: We first quantify the probability of SE being selected over MMM using the results of our frequency analysis, from Table 4. We refer the reader to Section 8 for all experimental details. Out of all paid workloads for which MMM and SE were selected (second and fourth rows of the table), we consider the probability for SE to be selected. SE is overwhelmingly more likely to be chosen over MMM on both BFS and DFS tasks, with the probability of selection being 90% for BFS and 95% for DFS. On the other hand, MMM is around twice as likely to be chosen over SE on both RRQ and IDEBench tasks, as the SE selection probability is 31% for RRQ and 34% for IDEBench. We conclude that the likelihood of the SE module being chosen, and thus the success of our heuristics, depends on the workload sequence since it influences the contents of our cache. Averaging across all four tasks, the likelihood of the SE module being chosen over MMM, is around 62%.

Theoretical Analysis: We analyze the conditions under which our heuristics result in SE module being selected. The SE module is only selected if it has a lower cost than the original strategy in MMM. The privacy cost for our mechanisms is inversely proportional to the error term, for a given (α, β) or α^2-expected total square error. Thus, our heuristics are only successful if they lead to the error term for the SE module to be smaller than the error term for MMM.

We recall Figure 5 included an example to show that expanding the strategy can lead to an increased error term. We analyze why such situations can arise and describe conditions for when our noise parameter-based heuristic can reduce the error. Our heuristic to choose $b_{\ell+1} < b_{\mathbf{P}}$ strictly improves the error under the following condition:

THEOREM D.1. *Given a workload* $\mathbf{W}$, *a strategy* $\mathbf{A}$ *of* ℓ *rows, and a noise vector* $\mathbf{b}$, *adding a new row to* $\mathbf{A}$ *to form* $\mathbf{A}_e$ *reduces the error, i.e.,*

$$\|\mathbf{WA}^+ Diag(\mathbf{b})\|_F^2 \geq \|\mathbf{WA}_e^+ Diag(\mathbf{b}||b_{\ell+1})\|_F^2 \quad (18)$$

if all entries in $\mathbf{b}$ *equal* b^* *and* $b_{\ell+1} < b_{\mathbf{P}} \leq b^*$, *for any* b^*.

PROOF. We recall the following theorem proved by Li et al. [18] in their MM paper.

$$\|\mathbf{WA}^+\|_F^2 \geq \|\mathbf{WA}_e^+\|_F^2 \quad (19)$$

For $\mathbf{b} = [b^* \cdots b^*]$, we transform Equation 18 to:

$$\begin{aligned} \|\mathbf{WA}^+ Diag(\mathbf{b})\|_F^2 &= \|\mathbf{WA}^+(b^*I)\|_F^2 = (b^*)^2\|\mathbf{WA}^+\|_F^2 & (20) \\ &\geq (b^*)^2\|\mathbf{WA}_e^+\|_F^2 & (21) \\ &= \|\mathbf{WA}_e^+ Diag(\mathbf{b}||b^*)\|_F^2 & (22) \\ &\geq \|\mathbf{WA}_e^+ Diag(\mathbf{b}||b_{\ell+1})\|_F^2 & (23) \end{aligned}$$

[2]https://github.com/cgebest/APEx
[3]https://git.uwaterloo.ca/m2mazmud/cachedp-public.git

where the first inequality comes from applying Li et al.'s result (Equation 19) and the final inequality by applying the condition: $b_{\ell+1} \leq b^*$. ☐

When the noise parameters are not all equal, the sufficiency conditions become more complicated. For example, if we modify the above counterexample to have a slightly more accurate expanded row ($b_{\ell+1} = 3b < 4b$), we get a lower error than MMM:

$$\|\mathbf{W}_1 \mathbf{A}_1^+ diag(\mathbf{b}_1)\| = 28b^2, \ \|\mathbf{W}_1 \mathbf{A}_{1e}^+ diag(\mathbf{b}_{1e})\| = 26.5b^2$$

Thus our greedy heuristic, which does not simply add entries less than b_p, but prioritizes the most accurate rows first, would be effective in this scenario. We also observe that distribution of the existing noise parameters in $\mathbf{b}$ is important. If the noise parameters are all close to each other, strategy expansion is likely to be more beneficial. For example changing the second entry of $\mathbf{b}_1$ from b to $2b$ (i.e. $\mathbf{b}_{1e} = [b, 2b, 5b, 4b]$), results in a smaller error than MMM:

$$\|\mathbf{W}_1 \mathbf{A}_1^+ diag(\mathbf{b}_1)\| = 31.0b^2, \ \|\mathbf{W}_1 \mathbf{A}_{1e}^+ diag(\mathbf{b}_{1e})\| = 30.2b^2$$

It is evident that both the distribution of the noise parameters in $\mathbf{b}$ and the new noise parameter $b_{\ell+1}$, influence the success of our accuracy heuristic. Additionally, we observe that a newly added row could satisfy our final heuristic by being a parent or a child of any of the existing rows in $\mathbf{A}_1$. We find that the structure of the expanded strategy $\mathbf{A}_{1e}$ also significantly influences the error term. For example, changing the final row in $\mathbf{A}_{1e}$ from $(1, 1, 1)$ to $(1, 0, 1)$ or $(0, 1, 1)$ also reduces the error for $\mathbf{A}_{1e}$ to $26.5b^2$ and $20b^2$ respectively, under the exact same noise parameters as our counterexample. However, changing the final row to $(1, 1, 0)$ increases the error to $34.7b^2$.

Conclusion: We observe that whether an additional row selected by our noise parameter heuristics will succeed in decreasing the error for SE over MMM, depends on how the new row changes elements in $\mathbf{W}\mathbf{A}_e^+$. However, we can guarantee that when the noise parameters in $\mathbf{b}$ are sufficiently similar, strategy expander will reduce the error regardless of the structure. Our structure-based heuristic only allows strategy queries related through a parent-child relationship to an existing strategy query in $\mathbf{A}$. Future research may model the success of this heuristic, by analyzing the relation between $\mathbf{W}\mathbf{A}^+$ and $\mathbf{W}\mathbf{A}_e^+$ for $\mathbf{A}$ and $\mathbf{A}_e$ that differ by a parent or child row.