Journal of Privacy and Confidentiality
Vol. 13 (1) 2023

Submitted Jun 2021
Published Aug 2023

HDMM: OPTIMIZING ERROR OF HIGH-DIMENSIONAL STATISTICAL QUERIES UNDER DIFFERENTIAL PRIVACY

RYAN MCKENNA, GEROME MIKLAU, MICHAEL HAY, AND ASHWIN MACHANAVAJJHALA

Unaffiliated
e-mail address: rmckenna21@gmail.com

College of Information & Computer Sciences, The University of Massachusetts, Amherst, MA 10002
e-mail address: miklau@cs.umass.edu

Department of Computer Science, Colgate University, Hamilton, NY 13346
e-mail address: mhay@colgate.edu

Department of Computer Science, Duke University, Durham, NC, 27708
e-mail address: ashwin@cs.duke.edu

Abstract. In this work we describe the High-Dimensional Matrix Mechanism (HDMM), a differentially private algorithm for answering a workload of predicate counting queries. HDMM represents query workloads using a compact implicit matrix representation and exploits this representation to efficiently optimize over (a subset of) the space of differentially private algorithms for one that is unbiased and answers the input query workload with low expected error. HDMM can be deployed for both ϵ-differential privacy (with Laplace noise) and (ϵ, δ)-differential privacy (with Gaussian noise), although the core techniques are slightly different for each. We demonstrate empirically that HDMM can efficiently answer queries with lower expected error than state-of-the-art techniques, and in some cases, it nearly matches existing lower bounds for the particular class of mechanisms we consider.

1. Introduction

Institutions like the U.S. Census Bureau and Medicare regularly release summary statistics about individuals, including population statistics cross-tabulated by demographic attributes [9, 39] and tables reporting on hospital discharges organized by medical condition and patient characteristics [25]. These data have the potential to reveal sensitive information, especially through joint analysis of multiple releases [20, 33, 45]. Differential privacy [14, 15] offers a framework for releasing statistical summaries of sensitive datasets, while providing formal and quantifiable privacy to the contributing individuals.

We consider the problem of *batch query answering* under differential privacy. That is, our goal is to release answers to a given query *workload*, consisting of a set of *predicate counting queries*, while satisfying differential privacy. A predicate counting query computes the number of individuals in the dataset who satisfy an arbitrary predicate ϕ (e.g., how many individuals have Income $\geq$ \$50,000). Workloads of predicate counting queries are

JPrivacy Confidentiality www.journalprivacyconfidentiality.org
DOI:10.29012/jpc.791

quite versatile as they are capable of expressing histograms, multi-dimensional range queries, group-by queries, data cubes, marginals, and arbitrary combinations thereof. Answering a batch of predicate counting queries has been widely studied by the research community. Past results have established theoretical lower bounds [5, 6, 8, 16, 22, 30, 38, 43] as well as a wealth of practical algorithms [2, 4, 10, 12, 24, 27–29, 31, 40–42, 42, 48–57].

One of the simplest mechanisms for answering a workload of queries is to add carefully calibrated Laplace or Gaussian noise directly to each of the workload query answers. The noise magnitude is calibrated to a property of the workload known as its *sensitivity*, which can be large for some workloads, resulting in a significant amount of noise. This method fails to adequately exploit structure in the workload and correlation amongst queries, and thus it often adds more noise than is strictly necessary to preserve differential privacy, resulting in suboptimal utility.

A better approach generalizes the basic noise addition mechanism by first **selecting** a new set of *strategy* queries, then **measuring** the strategy queries using a noise addition mechanism, and **reconstructing** answers to the workload queries from the noisy measurements of the strategy queries. Choosing an effective query answering strategy (different from the workload) can result in orders-of-magnitude lower error than the Laplace mechanism, with no cost to privacy. Many mechanisms for workload answering fall within the select-measure-reconstruct paradigm [2, 10, 12, 24, 27–29, 31, 31, 40–42, 48–54, 56, 57], differing primarily in the strategy selection step.

Example 1 (Hierarchical Strategy for Range Query Workloads)**.** *A canonical example of the select-measure-reconstruct paradigm is the binary hierarchical mechanism [24]. In essence, the hierarchical strategy includes hierarchically structured interval queries. This collection of queries has* $\log_2 n$ *sensitivity, where* n *is the size of the domain. Moreover, any range query can be answered by adding up answers to* $\log_2 n$ *hierarchical queries. This strategy works better than the two natural alternatives of (1) measuring the range queries directly, which has high sensitivity, and (2) measuring the histogram directly, which would have large reconstruction error.*

In general, we can characterize strategy selection as a search problem over a space of strategies, distinguishing prior work in terms of key algorithmic design choices: the search space, the cost function, and the type of search algorithm (greedy, local, global, etc.). These design choices impact the two key performance considerations: accuracy and scalability.

At one extreme are techniques that explore a narrow search space, making them efficient and scalable but not particularly accurate (in particular, their search space may include accurate strategies only for a limited class of workloads). For example, HB [41] generalizes the binary hierarchical strategy above by considering different branching factors. It performs a simple search to find the branching factor of the hierarchical strategy that minimizes an error measure that assumes the workload consists of all range queries (regardless of the actual input workload). It is efficient and can scale to higher dimensions, but it achieves competitive accuracy only when the workload consists of range queries and the data is low dimensional.

At the other extreme are techniques that search a large space, and adapt to the workload by finding a strategy within that space that offers low error on the workload, thereby making them capable of producing a more accurate strategy for the particular workload. However, this increased accuracy comes at the cost of high runtime and poor scalability. This is exemplified by the Matrix Mechanism [31]. The Matrix Mechanism represents the workload

and strategy as a matrix, and the data as a vector. With this representation, the select, measure, and reconstruct steps can be completely defined in the language of linear algebra. In addition, there is a simple formula for the expected error of any selected strategy matrix in terms of elementary matrix operations. This enables the Matrix Mechanism to select the optimal strategy (i.e., the one that offers least expected error) by solving a numerical optimization problem.

Using a matrix to represent a workload is appealing because the representation is expressive enough to capture an arbitrary collection of predicate counting queries, and it reveals any structure that may exist between the workload queries. However, the size of the workload matrix is equal to the number of queries times the size of the domain, and it is infeasible to represent large workloads defined over multi-dimensional domains as a matrix. Moreover, solving the optimization problem underlying strategy selection is nontrivial and expensive. In short, there is a need for new mechanisms that offer *generality* to a wide class of input workloads, *accuracy* on those workloads, and *scalability* to large multi-dimensional domains.

Overview of approach and contributions. This paper describes the High-Dimensional Matrix Mechanism (HDMM), which is a practical instantiation of the Matrix Mechanism (MM), capable of scaling to large multi-dimensional domains. HDMM offers the flexibility and workload-adaptivity of the Matrix Mechanism, while offering the scalability of simpler mechanisms. The three items listed below distinguish HDMM from the Matrix Mechanism:

- (Sections 5 and 7) The Matrix Mechanism represents query workloads *explicitly*, as fully materialized matrices, while HDMM uses a compact *implicit* matrix representation. This permits a lossless representation of queries that avoids a representation exponential in the number of attributes. The implicit representation consists of sub-workload matrices (usually one per attribute) which are used as terms in a Kronecker product. Further, we allow the workload to be expressed as unions of such Kronecker terms. This allows us to represent large multi-dimensional workloads efficiently while maintaining the key benefits that the explicit matrix representation offers.
- (Sections 4, 6 and 8) The numerical optimization problem at the heart of the Matrix Mechanism is practically infeasible, even for a single attribute with a domain of size 10. HDMM introduces four optimization routines for strategy selection: OPT_0, $\mathsf{OPT}_\otimes$, OPT_+, and OPT_M. OPT_0 is designed for *explicitly* represented workloads, and can scale to domains as large as 8192. $\mathsf{OPT}_\otimes$, OPT_+, and OPT_M are three different techniques for optimizing *implicitly* represented workloads (with implicitly represented strategies), and can scale to significantly larger domains. More concretely, $\mathsf{OPT}_\otimes$ and OPT_+ have linear dependence on the number of attributes, while OPT_0 and any method that deals with explicitly represented workloads has an exponential dependence on the number of attributes. See Table 2 for a more complete summary of the complexity of our proposed methods. These optimization routines differ in the space of strategies they consider. In all cases, the strategy search space is chosen so that is expressive enough to encode high-quality strategies, while also enabling tractable optimization.
- (Section 10) We also propose efficient algorithms for the measure and reconstruct steps of HDMM. In the Matrix Mechanism, these steps are implemented by performing matrix operations with the explicit workload and strategy matrices and the data vector. HDMM

exploits the implicit representation of the selected strategies to significantly speed up these steps.

As a result of these distinguishing items, HDMM overcomes the main scalability limitations of the Matrix Mechanism, and runs effectively on both low- and high-dimensional workloads.[1] In fact, in our experiments, we find it has higher accuracy than all prior select-measure-reconstruct techniques, even on input workloads for which the prior techniques were specifically designed. It also achieves reasonable runtime and scales more effectively than prior work that performs non-trivial optimization (see Section 11 for a detailed scalability evaluation). The main bottleneck of HDMM is *representing the data in vector form*, which requires space proportional to the domain size; HDMM can scale to domains as large as 10^9.[2] HDMM was first described by the authors in [34]. This paper provides a more complete description of HDMM and adds several new technical contributions, enumerated below:

(1) We generalize and extend HDMM to support (ϵ, δ)-differential privacy via Gaussian noise. This is a nontrivial extension: changing noise distributions fundamentally changes the optimization problems underlying MM and HDMM. We analyze this change and derive new optimization routines for strategy selection in this regime. All four of our core optimization routines $\mathsf{OPT}_0, \mathsf{OPT}_\otimes, \mathsf{OPT}_+$, and OPT_M require different changes to support Gaussian noise.
(2) We provide new results on the SVD bound [30], a simple formula which provides a lower bound on the achievable error of the Matrix Mechanism in terms of the properties of the workload. Specifically, we show how the SVD bound can be efficiently computed for implicitly represented workloads, and we use the SVD bound to provide additional theoretical justification for our optimization routines. We include the SVD bound in experiments to inform the optimality of the strategies found by our optimization routines.
(3) We provide a complete description of OPT_M, the optimization routine that searches over marginal query strategies. This is one of most important optimization routines because it generally produces the best strategies for marginal query workloads, one of the most common types of workloads for multi-dimensional data. In addition, we show that for marginal query workloads, it is sometimes possible to derive the optimal strategy in closed form.
(4) We provide a more comprehensive set of experiments, in both low-dimensional and high-dimensional settings, showing consistent utility improvements over other mechanisms. We also provide a detailed analysis of the experimental results.

Organization. This paper is organized as follows. In Section 2 we provide background on the data model and query representation. In Section 3 we provide background on differential privacy, including the Matrix Mechanism. In Section 4, we describe OPT_0, an optimization routine that approximately solves the Matrix Mechanism optimization problem for *explicitly* represented workloads. In Section 5, we show how many common workloads over high-dimensional domains can be *implicitly* represented in terms of Kronecker products. In Section 6, we describe $\mathsf{OPT}_\otimes$ and OPT_+: two optimization routines that can effectively optimize implicitly represented workloads. In Section 7, we show how marginal query

[1] We use the term dimensionality to refer to the size of the domain. In some other works, the dimensionality refers to the number of attributes in the domain. A high-dimensional domain in our sense of the word often arises from a handful of attributes, and is the setting HDMM is primarily designed for.

[2] We show in this paper that in certain special cases, we can bypass this fundamental limitation.

workloads can be represented implicitly, using an even more compact representation than the one given in Section 5. In Section 8, we describe $\mathsf{OPT}_{\mathsf{M}}$, an optimization routine that can efficiently optimize implicitly represented workloads with marginal query strategies. In Section 10, we describe the remaining steps of HDMM, including how to exploit the implicit representations to efficiently `MEASURE` the strategy queries and `RECONSTRUCT` the answers to the workload queries. We perform a thorough experimental study in Section 11. Related work is discussed in Section 12 and proofs are provided in the appendix.

2. Data model and query representation

In this section we introduce much of the notation and relevant background on the data model and query representation required to understand this work. We use as a running example, and motivating use case, the differentially private release of a collection of 10 tabulations from the 2010 *Summary File 1 (SF1)* [9], an important data product based on the Census of Population and Housing (CPH).

2.1. Notation.

A table of common notations is given in Table 1. In general, we adhere to the following conventions. Scalars and tuples are lowercase, non-bold. Sets are uppercase, non-bold. Vectors are lowercase, bold. Matrices are uppercase, bold. Implicit matrices are uppercase, blackboard bold.

2.2. Data and schema.

We assume a single-table relational schema $R(A_1 \dots A_d)$, where $attr(R)$ denotes the set of attributes of R. Subsets of attributes are denoted $\mathcal{A} \subseteq attr(R)$. Each attribute A_i has a finite domain $dom(A_i)$ with size $|dom(A_i)| = n_i$. The full domain of R is $dom(R) = dom(A_1) \times \cdots \times dom(A_d)$, and has size $n = \prod_i n_i$. An instance I of relation R is a multiset whose elements are tuples in $dom(R)$.

Example 2. *The* `Person` *relation has the following schema: six boolean attributes describing Race, two boolean attributes for Hispanic Ethnicity and Sex, Age in years between 0 and 114, and a Relationship-to-householder field that has 17 values. These queries are on a multidimensional domain of size* $2^6 \times 2 \times 2 \times 115 \times 17 = 500{,}480$. *The data also includes a geographic attribute encoding state (51 values including D.C.). The SF1+ queries are defined on a domain of size* $500{,}480 \times 51 = 25{,}524{,}480$.

Symbol	Meaning	Symbol	Meaning
A	Attribute	ϵ, δ	Privacy parameters
t	Tuple (database item)	$\mathcal{K}$	Noise addition mechanism
ϕ	Predicate	$\mathcal{K} = \mathcal{L}$	Laplace mechanism
Φ	Set of predicates	$\mathcal{K} = \mathcal{G}$	Gaussian mechanism
$\mathbb{I}$	Indicator function	$\|\boldsymbol{A}\|_{\mathcal{L}}$	L_1 sensitivity
$\mathcal{W}$	Logical workload	$\|\boldsymbol{A}\|_{\mathcal{G}}$	L_2 sensitivity
$\boldsymbol{x}$	Data vector	$\|\boldsymbol{A}\|_F$	Frobenius norm
$\boldsymbol{q}$	Query vector	$\otimes$	Kronecker product
$\boldsymbol{W}$	(Explicit) Workload matrix	$\mathbb{W}$	(Implicit) Workload matrix
$\boldsymbol{A}$	(Explicit) Strategy matrix	$\mathbb{A}$	(Implicit) Strategy matrix

Table 1. Table of notation.

2.3. **Logical view of queries.** Predicate counting queries are a versatile class, consisting of queries that count the number of tuples satisfying any logical predicate. We define below a natural logical representation of these queries, distinguished from a subsequent vector representation.

Definition 1 (Predicate counting query)**.** *A predicate on R is a boolean function $\phi : dom(R) \to \{0, 1\}$. A predicate can be used as a counting query on instance I of R whose answer is $\phi(I) = \sum_{t \in I} \phi(t)$.*

A predicate corresponds to a condition in the `WHERE` clause of an SQL statement, so in SQL a predicate counting query has the form: `SELECT Count(*) FROM R WHERE` ϕ.

When a predicate refers *only* to a subset of attributes $\mathcal{A} \subset attr(R)$ we may say that it is defined with respect to $\mathcal{A}$ and annotate it $\phi_{\mathcal{A}} : dom(\mathcal{A}) \to \{0, 1\}$. If $\phi_{\mathcal{A}}$ and $\phi_{\mathcal{B}}$ are predicates on attribute sets $\mathcal{A}$ and $\mathcal{B}$, then their conjunction is a predicate $\phi_{\mathcal{A}} \wedge \phi_{\mathcal{B}} : dom(\mathcal{A} \cup \mathcal{B}) \to \{0, 1\}$.

We assume that each query consists of *arbitrarily complex* predicates on each attribute, but require that they are combined across attributes with conjunctions. In other words, each ϕ is of the form $\phi = \phi_{A_1} \wedge \cdots \wedge \phi_{A_d}$. This facilitates the compact implicit representations described in Section 5. One approach to handling disjunctions (and other more complex query features) is to transform the schema by merging attributes. We illustrate this below in its application to the SF1 workload and provide a more general approach to disjunctive queries in Appendix A.

Example 3. *The SF1 workload consists of conjunctive conditions over its attributes, with the exception of conditions on the six binary race attributes, which can be complex disjunctions of conjunctions (such as "The number of Persons with two or more races"). We simply merge the six binary race attributes and treat it like a single $2^6 = 64$ size attribute (called simply* Race*). This schema transformation does not change the overall domain size, but allows every SF1 query to be expressed as a conjunction.*

2.4. **Logical view of query workloads.** A workload is a set of predicate counting queries $\Phi = \{\phi_1, \ldots, \phi_m\}$. A workload may consist of queries designed to support a variety of analyses or user needs, as is the case with the SF1 workload described above. Workloads may also be built from the sufficient statistics of models, or generated by tools that aid users in exploring data, or a combination thereof. For the privacy mechanisms considered here it is preferable for the workload to explicitly mention all queries of interest, rather than a subset of the queries that could act like a supporting view, from which the remaining queries of interest could be computed. Enumerating all queries of interest allows error to be optimized collectively. In addition, a workload query can be repeated, or equivalently, weighted, to express the preference for greater accuracy on that query.

Example 4. *Our example workload is a subset of queries from SF1 that can be written as predicate counting queries over a* `Person` *relation. (We omit other queries in SF1 that involve groups of persons organized into households; for brevity we refer to our selected queries simply as SF1.) Our SF1 workload has 4151 predicate counting queries, each of the form* `SELECT Count(*) FROM Person WHERE` ϕ*, where ϕ specifies some combination of demographic properties (e.g. number of Persons who are Male, over 18, and Hispanic) and thus each query reports a count at the national level.*

Example 5. *A workload we call SF1+ consists of the national level queries in SF1* as well as *the same queries at the state level for each of 51 states. We can succinctly express the state level queries as an additional 4151 queries of the form:* `SELECT state, Count(*) FROM` Person `WHERE` ϕ `GROUP BY state`. *Thus, SF1+ can be represented by a total of* $4151 + 4151 = 8302$ *SQL queries. (The* `GROUP BY` *is a succinct way to represent a potentially large set of predicate counting queries.) The SF1+ queries are defined on a domain of size* $500{,}480 \times 51 = 25{,}524{,}480$. *In addition to their SQL representation, the SF1 and SF1+ workloads can be naturally expressed in a logical form defined in Section 5.1. We use* $\mathcal{W}_{SF1}$ *and* $\mathcal{W}_{SF1+}$ *to denote the logical forms of SF1 and SF1+ respectively.*

Structured multi-dimensional workloads. Multi-dimensional workloads are often defined in a structured form, as *products* and *unions of products*, that we will exploit later in our implicit representations. Following the notation above, we write $\Phi_{\mathcal{A}}$ to denote a set of predicates, each mentioning only attributes in $\mathcal{A}$. For example, the following are common predicate sets defined over a single attribute A of tuple t:

I	Identity_A	$= \{\, \mathbb{I}[t_A = a] \mid a \in dom(A) \,\}$
P	Prefix_A	$= \{\, \mathbb{I}[t_A \leq a] \mid a \in dom(A) \,\}$
R	AllRange_A	$= \{\, \mathbb{I}[a \leq t_A \leq b] \mid a, b \in dom(A), a \leq b \,\}$
T	Total_A	$= \{\, \mathbb{I}[\text{True}] \,\}$

Above, $\mathbb{I}$ is the indicator function, e.g., $\mathbb{I}[t_A = a] = \phi(t_A) = \begin{cases} 1 & \text{if } t_A = a \\ 0 & \text{otherwise} \end{cases}$.

Identity_A contains one predicate for each element of the domain. Both Prefix_A and Range_A rely on an ordered $dom(A)$; they contain predicates defining a CDF (i.e., sufficient to compute the empirical cumulative distribution function), and the set of all range queries, respectively. The predicate set Total_A, consists of a single predicate, returning *True* for any $a \in dom(A)$, and thus counting all records.

We can construct multi-attribute workloads by taking the cross-product of predicate sets defined for single attributes, and *conjunctively* combining individual queries.

Definition 2 (Product)**.** *The product of two predicate sets* $\Phi_{\mathcal{A}}$ *and* $\Phi_{\mathcal{B}}$ *is another predicate set* $\Phi_{\mathcal{A}} \times \Phi_{\mathcal{B}} = \{\phi_{\mathcal{A}} \wedge \phi_{\mathcal{B}} \mid \phi_{\mathcal{A}} \in \Phi_{\mathcal{A}}, \phi_{\mathcal{B}} \in \Phi_{\mathcal{B}}\}$ *containing the conjunction of every pair of predicates.*

We describe several examples of workloads constructed from products and unions of products below.

Example 6 (Single query as product)**.** *A predicate counting query in the SF1 workload is:* `SELECT Count(*) FROM` Person `WHERE sex=M AND age < 5`. *We can express this query as a product: first, define predicate set* $\Phi_1 = \{\mathbb{I}[t_{sex} = M]\}$ *and predicate set* $\Phi_2 = \{\mathbb{I}[t_{age} < 5]\}$. *The query is expressed as the product* $\Phi_1 \times \Phi_2$. *(We omit Total on the other attributes for brevity.)*

Example 7 (`GROUP BY` query as product)**.** *A* `GROUP BY` *query can be expressed as a product by including an Identity predicate set for each grouping attribute and a singleton predicate set for each attribute in the* `WHERE` *clause. The product would also include Total for each attribute not mentioned in the query. For example, the query* `SELECT sex, age, Count(*) FROM` Person `WHERE hispanic = TRUE GROUP BY sex, age` *is expressed as* $I_{sex} \times I_{age} \times \Phi_3$

where $\Phi_3 = \{\mathbb{I}[t_{hispanic} = True]\}$. *This product contains* 2×115 *counting queries, one for each possible setting of sex and age.*

Example 8 (Marginal and Prefix-Marginal)**.** *A marginal query workload is defined by the product of one or more Identity predicates on selected attributes and Total on all other attributes. For example,* $I_{\texttt{Sex}} \times I_{\texttt{Age}} \times I_{\texttt{Hispanic}}$ *contains predicates to compute one three-way marginal, and consists of* $2 \times 115 \times 2$ *counting queries, one for each possible setting of Sex, Age and Hispanic. This is equivalent to a* `GROUP BY` *query on Sex, Age, and Hispanic with no* `WHERE` *clause.*

A prefix-marginal query workload is a natural generalization of a marginal query workload which can be obtained by replacing one or more of the Identity predicates with Prefix. For example, $I_{\texttt{Sex}} \times P_{\texttt{Age}} \times I_{\texttt{HIspanic}}$ *contains predicates to compute one three-way prefix-marginal, and consists of* $2 \times 115 \times 2$ *counting queries of the form* $\mathbb{I}[t_{sex} = a, t_{age} \leq b, t_{hispanic} = c]$.

Marginals query workloads are very common workloads that make sense for domains with categorical attributes, while prefix-marginals are more natural for domains with both categorical and discretized numeric attributes.

Example 9 (SF1 Tabulation as Product)**.** *Except for the population total, the 46 queries in the P12 tabulation of the Census SF1 workload [9] can be described by a single product:* $I_{\texttt{Sex}} \times R_{\texttt{Age}}$ *where* $R_{\texttt{Age}}$ *is a particular set of range queries including* $[0, 114], [0, 4], [5, 9], [10, 14], \ldots [85, 114]$.

Unions of products. Our workloads often combine multiple products as a union of the sets of queries in each product. For example, the queries required to compute all three-way marginals could be represented as a union of $\binom{d}{3}$ workloads, each a product of the Identity predicate set applied to three attributes. The input to the algorithms that follow is a logical workload consisting of a union of products, each representing one or possibly many queries.

Definition 3 (Logical workload)**.** *A logical workload* $\mathcal{W} = \{Q_1 \ldots Q_k\}$ *consists of a set of products* Q_i *where each* $Q_i = \Phi^{(i)}_{A_1} \times \cdots \times \Phi^{(i)}_{A_d}$.

Example 10 (SF1 as union of products)**.** *The SF1 workload can be represented in a logical form, denoted* $\mathcal{W}_{SF1}$*, that consists of a union of* $k = 4151$ *products, each representing a single query. Because these queries are at the national level, there is a Total predicate set on the State attribute. The logical form of the SF1+ workload, denoted* $\mathcal{W}_{SF1+}$*, includes those products, plus an additional 4151 products that are identical except for replacing the Total on State with an Identity predicate set. There are a total of* $k = 8302$ *products, representing a total of* $4151 + (51 \times 4151) = 215{,}852$ *predicate counting queries. While this is a direct translation from the SQL form, there are alternative representations that are more compact. First, we can reduce to* $k = 4151$ *products by simply adding* $True$ *to the Identity predicate set on State to capture the national counts. Furthermore, through manual inspection, we found that both* $\mathcal{W}_{SF1}$ *and* $\mathcal{W}_{SF1+}$ *can be even more compactly represented as the union of 32 products—we use* $\mathcal{W}^*_{SF1}$ *and* $\mathcal{W}^*_{SF1+}$ *to denote more compact logical forms. This reduction was accomplished by carefully analyzing the structure of the queries, and combining groups of related queries into single products where possible, as in Example 9. This results in significant space savings (as described shortly in Example 14) and runtime improvements.*

2.5. **Explicit data and query vectorization.** The vector representation of predicate counting queries (and the data they are evaluated on) is central to the select-measure-reconstruct paradigm. The vector representation of instance I is denoted $\boldsymbol{x}_I$ (or simply $\boldsymbol{x}$ if the context is clear) and called the *data vector* or *histogram*.

Definition 4 (Data vector)**.** *The data vector representation of an instance I, denoted $\boldsymbol{x}_I$, is a vector indexed by tuples $t \in dom(R)$, so that $\boldsymbol{x}_I(t) = \sum_{t' \in dom(R)} \mathbb{I}[t = t']$.*

Informally, $\boldsymbol{x}(t)$ counts the number of occurences of t in I. Note that, throughout the paper, the representation of the data vector is *always* explicit; it is the representation of queries that will be implicit. Every predicate counting query ϕ also has a corresponding vector form.

Definition 5 (Vectorized query)**.** *The vector representation of a predicate counting query ϕ, denoted $\boldsymbol{q}_\phi = vec(\phi)$ is a vector indexed by tuples $t \in dom(R)$, so that $\boldsymbol{q}_\phi(t) = \phi(t)$.*

The function $vec(\phi)$ which transforms a logical predicate into its corresponding vector form has a simple implementation: compute $\phi(t)$ for every $t \in dom(R)$ and store the results in a vector. Note that both the data vector and the vectorized query have size $|dom(R)| = N$.

Proposition 1 (Query evaluation)**.** *A predicate counting query can be answered by computing the dot product between the query vector and data vector: $\phi(I) = \boldsymbol{q}_\phi^\top \boldsymbol{x}_I$.*

To see why this works, observe that $\phi(I) = \sum_{t \in I} \phi(t) = \sum_{t \in dom(R)} \phi(t) \boldsymbol{x}_I(t) = \boldsymbol{q}_\phi^\top \boldsymbol{x}_I$. A single predicate counting query is represented as a vector, so a workload of predicate counting queries can be represented as a matrix in which queries are rows. For logical workload $\mathcal{W}$, its (explicit) matrix form is written $\boldsymbol{W}$, and the evaluation of the workload is equivalent to the matrix product $\boldsymbol{W}\boldsymbol{x}_I$. Note that the size of the workload matrix is $m \times n$ where m is the number of queries, $\boldsymbol{x}_I$ is $n \times 1$, and the vector of workload answers is $m \times 1$.[3]

3. Privacy background

Differential privacy is a property of a randomized algorithm that bounds the difference in output distribution induced by changes to an individual's data. Let $nbrs(I)$ be the set of databases differing from I in at most one record.

Definition 6 (Differential Privacy [14])**.** *A randomized algorithm $\mathcal{K}$ is (ϵ, δ)-differentially private if for any instance I, any $I' \in nbrs(I)$, and any subset of measurable outputs $O \subseteq Range(\mathcal{K})$,*

$$Pr[\mathcal{K}(I) \in O] \leq \exp(\epsilon) \times Pr[\mathcal{K}(I') \in O] + \delta.$$

When $\delta = 0$, we say $\mathcal{K}$ is ϵ-differentially private. In this work we consider algorithms for answering a workload of predicate counting queries, in which case we invoke the mechanism with the workload matrix and data vector: $\mathcal{K}(\boldsymbol{W}, \boldsymbol{x})$. We focus on mechanisms that give unbiased answers to the workload queries, which we simply refer to as data-indendent mechanisms in this paper. Below we define two simple mechanisms of this form: the Laplace mechanism ($\mathcal{K} = \mathcal{L}$) and the Gaussian mechanism ($\mathcal{K} = \mathcal{G}$).

[3]All vectors in this work are assumed to be column vectors. With some abuse of notation, we will say a $n \times 1$ column vector is simply a size n vector.

Definition 7 (Laplace mechanism)**.** *Given an $m \times n$ query matrix $\boldsymbol{W}$, and a noise magnitude σ, the Laplace Mechanism $\mathcal{L}$ outputs the vector: $\mathcal{L}(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \|\boldsymbol{W}\|_{\mathcal{L}} Lap(b)^m$ where $Lap(b)^m$ is a vector of m independent samples from a Laplace distribution with scale b, and $\|\boldsymbol{W}\|_{\mathcal{L}} = \|\boldsymbol{W}\|_{1\to 1}$ denotes the maximum L_1 norm of the columns of $\boldsymbol{W}$.*

Definition 8 (Gaussian mechanism)**.** *Given an $m \times n$ query matrix $\boldsymbol{W}$, and a noise magnitude σ, the Gaussian mechanism $\mathcal{G}$ outputs the vector: $\mathcal{G}(\boldsymbol{A}, \boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \|\boldsymbol{W}\|_{\mathcal{G}} Gaus(\sigma)^m$ where $Gaus(\sigma)^m$ is a vector of m independent samples from a Gaussian distribution with scale σ and $\|\boldsymbol{W}\|_{\mathcal{G}} = \|\boldsymbol{W}\|_{1\to 2}$ denotes the maximum L_2 norm of the columns of $\boldsymbol{W}$.*

The quantities $\|\boldsymbol{W}\|_{\mathcal{L}}$ and $\|\boldsymbol{W}\|_{\mathcal{G}}$ above are the L_1 *sensitivity* and L_2 *sensitivity* of the query set defined by $\boldsymbol{W}$ respectively, since they measure the maximum difference in the answers to the queries in $\boldsymbol{W}$ on any two databases that differ only by a single record [31]. In the remainder of the paper, we will use $\|\boldsymbol{W}\|_{\mathcal{K}}$ to denote this sensitivity norm when referring to a general quantity that applies for both Laplace noise ($\mathcal{K} = \mathcal{L}$) and Gaussian noise ($\mathcal{K} = \mathcal{G}$). As long as b or σ is sufficiently large, these two mechanisms can provide differential privacy. The precise conditions are stated in Proposition 2.

Proposition 2 (Privacy of Laplace and Gaussian mechanisms [3, 15])**.** *The Laplace mechanism is ϵ-differentially private as long as $b \geq \frac{1}{\epsilon}$ and the Gaussian mechanism is (ϵ, δ) differentially private as long as $\delta \geq \Phi(\frac{1}{2\sigma} - \epsilon\sigma) - \exp(\epsilon)\Phi(-\frac{1}{2\sigma} - \epsilon\sigma)$, where Φ is the cumulative distribution function of the standard Gaussian distribution.*

Above, the condition on σ for the Gaussian mechanism corresponds to the so-called "analytic Gaussian mechanism" [3], which *exactly* calibrates the minimum σ needed to ensure (ϵ, δ)-differential privacy. For this method, σ can be obtained numerically with a root-finding algorithm. Other formulas exist for σ using a classical analysis of the Gaussian mechanism [15] or through Rényi differential privacy or concentrated differential privacy [7, 36], although these expressions tend to overestimate the noise magnitude required to achieve (ϵ, δ)-DP. It is straightforward to analyze the error of these two mechanisms since they add i.i.d. noise with the same (known) variance to all workload queries. In particular, we use expected total squared error (TSE) as our error metric:

Definition 9 (Expected Error [28])**.** *Given a $m \times n$ workload matrix $\boldsymbol{W}$ and a differentially-private algorithm $\mathcal{K}$, the expected total squared error is,*

$$TSE(\boldsymbol{W}, \mathcal{K}) = \max_{\boldsymbol{x}} \mathbb{E}[\|\boldsymbol{W}\boldsymbol{x} - \mathcal{K}(\boldsymbol{W}, \boldsymbol{x})\|_2^2],$$

where the expectation is taken over the randomness in the privacy mechanism $\mathcal{K}$.

For unbiased mechanisms which produce correct answers in expectation, the TSE is the same for all data vectors $\boldsymbol{x}$. While there are many other error metrics one could use, we focus on TSE in this work, since our goal is to develop a scalable instantiation of the matrix mechanism, which also targets TSE. A primary reason that TSE is appealing is because it is analytically easier to work with than alternatives like L_1 and L_∞ error metrics. Ultimately, the right error metric to target is application dependent. As TSE is our main focus in this paper, we will simply use the term "expected error" when referring to it. As we show below, we can readily reason about the expected error of the Laplace and Gaussian mechanisms:

Proposition 3 (Error of Laplace and Gaussian mechanisms)**.** *The Laplace and Gaussian mechanisms are unbiased and have the following expected error:*

$$TSE(\boldsymbol{W}, \mathcal{L}) = 2mb^2 \|\boldsymbol{W}\|_{\mathcal{L}}^2, \qquad TSE(\boldsymbol{W}, \mathcal{G}) = m\sigma^2 \|\boldsymbol{W}\|_{\mathcal{G}}^2.$$

As evident by Proposition 3, the expected error of these mechanisms depends crucially on the sensitivity of the query matrix $\boldsymbol{W}$, and if this is large then the total squared error will also be large. We now introduce a generalization of these mechanisms that often has better expected error.

3.1. **The matrix mechanism.** The core idea of the matrix mechanism is to apply the mechanism $\mathcal{K}$ on a new query matrix $\boldsymbol{A}$, then use the noisy answers to the queries in $\boldsymbol{A}$ to estimate answers to the queries in $\boldsymbol{W}$. The benefits of this approach will become clear when we reason about the expected error.

Definition 10 (Matrix mechanism [28])**.** *Given a $m \times n$ workload matrix $\boldsymbol{W}$, a $p \times n$ strategy matrix $\boldsymbol{A}$, and a differentially private algorithm $\mathcal{K}(\boldsymbol{A}, \boldsymbol{x})$ that answers $\boldsymbol{A}$ on $\boldsymbol{x}$, the mechanism $\mathcal{M}_{\boldsymbol{A},\mathcal{K}}$ outputs the following vector: $\mathcal{M}_{\boldsymbol{A},\mathcal{K}}(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{W}\boldsymbol{A}^+\mathcal{K}(\boldsymbol{A}, \boldsymbol{x})$.*

The privacy of the Matrix mechanism follows from the privacy of $\mathcal{K}$, since that is the only part of the mechanism that has access to the true data. In this paper, we assume $\mathcal{K}$ is either the Laplace mechanism or the Gaussian mechanism, although in principle other noise-addition mechanisms are also possible [18, 22, 31]. Under some mild conditions stated below, the Matrix mechanism is unbiased and the error can be expressed analytically as shown below:

Proposition 4 (Error of the Matrix mechanism [28])**.** *The matrix mechanism is unbiased, i.e., $\mathbb{E}[\mathcal{M}_{\boldsymbol{A},\mathcal{K}}(\boldsymbol{W}, \boldsymbol{x})] = \boldsymbol{W}\boldsymbol{x}$, and has the following expected error:*

$$\begin{aligned} TSE(\boldsymbol{W}, \mathcal{M}_{\boldsymbol{A},\mathcal{K}}) &= TSE(\boldsymbol{A}, \mathcal{K}) \left\|\boldsymbol{W}\boldsymbol{A}^+\right\|_F^2 \\ &\propto \|\boldsymbol{A}\|_{\mathcal{K}}^2 \left\|\boldsymbol{W}\boldsymbol{A}^+\right\|_F^2, \end{aligned}$$

as long as $\boldsymbol{W}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{W}$ and $\mathcal{K}$ adds i.i.d. noise with mean 0.

When invoked with $\boldsymbol{A} = \boldsymbol{W}$, the matrix mechanism is very similar to base mechanism $\mathcal{K}$, but the error is typically lower. The term $\|\boldsymbol{W}\boldsymbol{W}^+\|_F^2$ in the error formula is equal to the rank of $\boldsymbol{W}$, which is bounded above by m. This implies that the error of $\mathcal{M}_{\boldsymbol{W},\mathcal{K}}$ can never be higher than $\mathcal{K}$. Further, there are often much better strategies to select than $\boldsymbol{A} = \boldsymbol{W}$.

Finding the best strategy $\boldsymbol{A}$ for a given workload $\boldsymbol{W}$ is the main technical challenge of the matrix mechanism. This strategy selection problem can be formulated as a constrained optimization problem. However, solving this problem is computationally expensive, especially when $\mathcal{K} = \mathcal{L}$, where it is generally infeasible to solve it for any nontrivial input workload.

3.2. **Lower bounds on error.** An important theoretical question is to identify, or bound, how low the error of the matrix mechanism can be for a given workload $\boldsymbol{W}$. This is useful because finding the strategy with minimum error is a difficult (and often intractable) problem, but computing a lower bound on error can be done efficiently. Knowing how low the error can be allows one to compare the error of a concrete strategy to the lower bound to see how close to optimal it is. Additionally, the lower bound can be used to make important policy decisions, such as setting the privacy budget, or whether it is worth investing the resources to find a good strategy (as opposed to using other types of mechanisms like data-dependent ones). Li and Miklau studied this problem and derived the SVD bound [30].

Definition 11 (SVD Bound [30]). *Given a $m \times n$ workload $\boldsymbol{W}$, the singular value bound is:*

$$SVDB(\boldsymbol{W}) = \frac{1}{n}\left(\lambda_1 + \cdots + \lambda_n\right)^2,$$

where $\lambda_1, \ldots, \lambda_n$ are the singular values of $\boldsymbol{W}$.

The SVD bound gives a lower bound on the error achievable by the matrix mechanism.

Proposition 5 (SVD Bound [30]). *Given a $m \times n$ workload $\boldsymbol{W}$ and a $p \times n$ strategy $\boldsymbol{A}$ that supports $\boldsymbol{W}$:*

$$SVDB(\boldsymbol{W}) \leq \|\boldsymbol{A}\|_{\mathcal{K}}^2 \left\|\boldsymbol{W}\boldsymbol{A}^+\right\|_F^2.$$

The SVD Bound is known to be tight when $\mathcal{K} = \mathcal{G}$ and some conditions on $\boldsymbol{W}$ are satisfied, meaning there is some strategy $\boldsymbol{A}$ that achieves the equality. When $\mathcal{K} = \mathcal{L}$ the bound may not be tight however. In this paper we use the SVD Bound to evaluate the quality of the strategies found by our optimization routines. We also derive expressions for efficiently computing the SVD Bound for implicitly represented workloads, and use this analysis to theoretically justify our optimization routines. Other results exist that bound the error of any data-independent mechanism [16].

4. Optimizing Explicit Workloads

In this section, we introduce the main optimization problem that underlies strategy selection for the matrix mechanism, and describe OPT_0, our algorithm for solving it. We assume for now that the workload is represented *explicitly* as a dense matrix. The methods we describe are useful by themselves for workloads defined over modest domains (namely, those smaller than about $n = 10^4$), and they are an essential building block for the more scalable methods we describe in Section 6.

4.1. The optimization problem.

Our goal is to find a query strategy that offers minimal expected error on the workload. Using the analytic error formula from Proposition 4, this can be defined as a constrained optimization problem. One formulation of this problem is stated below.

Problem 1 (Matrix Mechanism Optimization [31]). *Given an $m \times n$ workload matrix $\boldsymbol{W}$:*

$$\begin{aligned} \underset{\boldsymbol{A}}{\textit{minimize}} \quad & \|\boldsymbol{A}\|_{\mathcal{K}}^2 \left\|\boldsymbol{W}\boldsymbol{A}^+\right\|_F^2 \\ \textit{subject to} \quad & \boldsymbol{W}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{W} \end{aligned} \tag{4.1}$$

For a number of reasons, this optimization problem is difficult to solve exactly: it has many variables, it is not convex, and both the objective function and constraints involve $\boldsymbol{A}^+$, which can be slow to compute. In addition, $\|\boldsymbol{W}\boldsymbol{A}^+\|_F^2$ has points of discontinuity near the boundary of the constraint $\boldsymbol{W}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{W}$. This problem was originally formulated as a rank-constrained semi-definite program [28], and, while algorithms exist to find the global optimum, they require $O(m^4(m^4 + n^4))$ time, making it infeasible in practice.

Gradient-based numerical optimization techniques can be used to find locally optimal solutions to Problem 1. These techniques begin by guessing a solution $\boldsymbol{A}_0$ and then iteratively improve it using the gradient of the objective function to guide the search. The process ends after a number of iterations are performed, controlled by a stopping condition based on improvement of the objective function. The constraints complicate the problem further, but

even if we ignore them, gradient-based optimization is slow, as the cost of computing the objective function for general $\boldsymbol{A}$ is $O(n^3)$, e.g., requiring more than 6 minutes for $n = 8192$.

In the next subsections, we provide algorithms for efficiently solving Problem 1. We separately consider the two cases of Gaussian noise and Laplace noise, as the required techniques are quite different.

4.2. **Strategy Optimization with Gaussian Noise.** While Problem 1 with $\mathcal{K} = \mathcal{G}$ is not convex in its current form, it can be reformulated into an equivalent problem that is convex [31, 53]. The key idea is that the objective function can be expressed in terms of $\boldsymbol{X} = \boldsymbol{A}^\top \boldsymbol{A}$, since $\|\boldsymbol{A}\|_{\mathcal{G}}^2 = max(diag(\boldsymbol{A}^\top \boldsymbol{A}))$ and $\|\boldsymbol{W}\boldsymbol{A}^+\|_F^2 = tr[(\boldsymbol{A}^\top \boldsymbol{A})^+(\boldsymbol{W}^\top \boldsymbol{W})]$. This allows us to optimize $\boldsymbol{X}$ instead of $\boldsymbol{A}$, and then we can recover $\boldsymbol{A}$ by performing Cholesky decomposition on $\boldsymbol{X}$. Remarkably, the resulting problem is convex with respect to $\boldsymbol{X}$.

Definition 12 (Convex Reformulation [53])**.** *Given a workload matrix $\boldsymbol{W}$ of rank n, let $\mathsf{OPT}_0(\boldsymbol{W}) = \boldsymbol{A}$ where $\boldsymbol{A}^\top \boldsymbol{A}$ is a Cholesky decomposition of $\boldsymbol{X}^*$ and:*

$$\begin{aligned} \boldsymbol{X}^* = \underset{\boldsymbol{X}}{minimize} \quad & tr[\boldsymbol{X}^{-1}(\boldsymbol{W}^\top \boldsymbol{W})] \\ subject\ to \quad & diag(\boldsymbol{X}) = \boldsymbol{1} \\ & \boldsymbol{X} \succ 0 \end{aligned} \tag{4.2}$$

While the above problem is convex, it is still nontrivial to solve due to the dependence on the matrix inverse and the constraint $\boldsymbol{X} \succ 0$ ($\boldsymbol{X}$ is positive definite). The equality constraint $diag(\boldsymbol{X}) = 1$ and corresponding optimization variables $diag(\boldsymbol{X})$ can easily be eliminated from the problem since they must always equal 1. Additionally $\boldsymbol{X}$ must be a symmetric matrix so we can optimize over the lower triangular entries, essentially reducing the number of optimization variables by a factor of two. The full details of a conjugate gradient algorithm for solving this problem are available in [53]. The per-iteration runtime of their "COA" algorithm is $O(n^3)$, and it typically requires about 50 iterations to converge. In practice it is able to scale up to about $n \approx 10^4$. The algorithm works well in practice when n is small, but is not particularly robust for some workloads when n is larger, which we observe empirically in Section 11 (e.g., COA on prefix workload in Table 5).

We thus design our own algorithm to solve the same optimization problem, which is based on the same principles as the COA technique, but is more robust in practice. There are two key differences in our implementation. First, we initialize the optimization intelligently by setting $\boldsymbol{X} = \boldsymbol{P}\sqrt{\boldsymbol{\Lambda}}\boldsymbol{P}^\top$ where $\boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^\top$ is the eigen-decomposition of $\boldsymbol{W}^\top \boldsymbol{W}$. This an approximation to the optimal strategy based on the SVD bound [30], and acts as a very good initialization. Second, instead of using the custom-designed conjugate gradient algorithm proposed in [53], we simply use scipy.optimize, an off-the-shelf optimizer. We heuristically ignore the constraint $\boldsymbol{X} \succ 0$ during optimization, using a large loss value when it is not satisfied. This makes the problem unconstrained, and readily solvable by scipy.optimize. The constraint is verified to hold at the end of the optimization. These changes lead to more robust optimization that produce strategies nearly matching the SVD bound, as we show experimentally in Section 11.

There are other reformulations of Problem 1 that have been studied in the context of Gaussian noise including [11, 16], and these formulations lead to different algorithms for strategy optimization. Other related problem formulations have been proposed and studied, including [37, 38].

4.3. **Strategy Optimization with Laplace noise.** Unfortunately, the techniques used in the previous section do not apply to the Laplace noise setting, as the sensitivity norm $\|\boldsymbol{A}\|_{\mathcal{L}}$ cannot be expressed in terms of $\boldsymbol{A}^\top\boldsymbol{A}$. In this section, we describe an alternate approach to approximately solve Problem 1: parameterized strategies. Our key idea is to judiciously restrict the search space of the optimization problem to simplify the optimization while retaining expressivity of the search space. While our approach does not necessarily produce a globally optimal solution to Problem 1, with good parameterizations it can still find state-of-the-art strategies. Below we describe the idea of parameterized strategies in its full generality. Then we propose a specific parameterization that works well for a variety of input workloads.

A parameterization is a function $\boldsymbol{A}(\boldsymbol{\theta})$ mapping a real-valued parameter vector $\boldsymbol{\theta}$ to a strategy $\boldsymbol{A}$. Optimizing over a parameterized strategy space can be performed by optimizing $\boldsymbol{\theta}$ rather than $\boldsymbol{A}$. In the extreme case, there may be one entry in $\boldsymbol{\theta}$ for every entry in $\boldsymbol{A}$, but a more careful design of the parameterization with fewer parameters and a smart mapping between the entries of the parameter vector and the entries of the strategy matrix can lead to more effective optimization. There are several design considerations for setting up a good parameterization. First, the parameterization must be expressive enough to encode high-quality strategies (this may depend on the workload). Second, the parameterization should have structure that makes the optimization problem more computationally tractable (such as eliminating constraints). Third, the parameterization may encode domain expertise about what a good strategy should look like, which could make it easier to find high-quality local minima.

Several existing privacy mechanisms can be thought of as an instance of this parameterization framework [12, 27, 29, 41, 54]. These mechanisms typically are designed for a specific workload or workload class. For example, Qardaji et al. [41] consider the space of hierarchical strategies, which is parameterized by a single parameter: the branching factor of the hierarchy, which is chosen to minimize MSE on the workload of *all range queries*. Li et al. [27] consider the space of weighted hierarchical strategies (with fixed branching factor), which is parameterized by vector of scaling factors for each query in the hierarchy. This method adapts to the input workload, but the hierarchical parameterization only works well for workloads of range queries. Ding et al. [12] consider the space of marginal query strategies (and workloads of the same form), the parameters can be thought of as a binary vector of length 2^d corresponding to which marginals should be included in the strategy. These parameters are optimized using a greedy heuristic. Li et al. [29] consider the space of strategies containing the eigenvectors of the workload gram matrix, which is parameterized by a vector of scaling factors for each eigenvector. This parameterization works well when $\mathcal{K} = \mathcal{G}$, but not when $\mathcal{K} = \mathcal{L}$, and it has recently been subsumed by the work of Yuan et al. [53] for $\mathcal{K} = \mathcal{G}$.

We now present a new general-purpose parameterization, called p-Identity, which handles these considerations without making strict assumptions about the structure of the workload. It also out-performs all of the existing parameterizations, even on the workloads for which they were designed. The parameters of a p-Identity strategy are more naturally interpreted as a matrix $\boldsymbol{\Theta}$ rather than a vector $\boldsymbol{\theta}$ so we instead use the notation $\boldsymbol{A}(\boldsymbol{\Theta})$.

Definition 13 (p-Identity strategies)**.** *Given a $p \times n$ matrix of non-negative values $\boldsymbol{\Theta}$, the p-Identity strategy matrix $\boldsymbol{A}(\boldsymbol{\Theta})$ is defined as follows:*

$$A(\boldsymbol{\Theta}) = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{\Theta} \end{bmatrix} \boldsymbol{D},$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{D} = diag(1 + \mathbf{1}^\top \boldsymbol{\Theta})^{-1}$.

Intuitively, p-Identity strategies encode $n + p$ queries, including n weighted identity queries that count the number of records in the database for each domain element, as well as p arbitrary linear queries determined by $\boldsymbol{\Theta}$. The diagonal matrix $\boldsymbol{D}$ is used to re-weight the strategy so that $\|\boldsymbol{A}\|_{\mathcal{L}} = 1$ and each column of $\boldsymbol{A}$ has the same L_1 norm. This is an example of domain knowledge incorporated into the parameterization, as it has been shown that optimal strategies have uniform column norm [31].[4]

Example 11. *For $p = 2$ and $n = 3$, we illustrate below how $\boldsymbol{A}(\boldsymbol{\Theta})$ is related to its parameter matrix, $\boldsymbol{\Theta}$:*

$$\boldsymbol{\Theta} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix}, \qquad \boldsymbol{A}(\boldsymbol{\Theta}) = \begin{bmatrix} 0.33 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.2 \\ 0.33 & 0.5 & 0.6 \\ 0.33 & 0.25 & 0.2 \end{bmatrix}.$$

For this class of parameterized strategies, the resulting optimization problem requires optimizing $\boldsymbol{\Theta}$ instead of $\boldsymbol{A}$ and is stated below; we use OPT_0 to denote the operator that solves this problem.

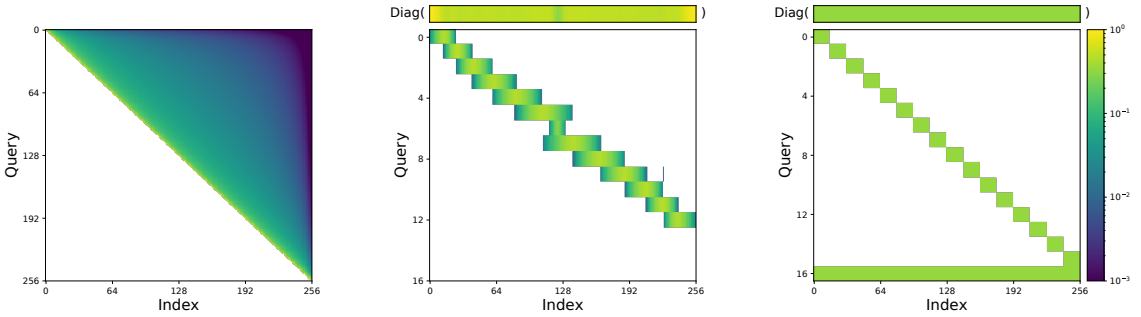
Definition 14 (parameterized optimization)**.** *Given a workload matrix $\boldsymbol{W}$ and hyper-parameter p, let $\mathsf{OPT}_0(\boldsymbol{W}) = \boldsymbol{A}(\boldsymbol{\Theta}^*)$ where:*

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta} \in \mathbb{R}_+^{p \times n}}{argmin} \quad \left\| \boldsymbol{W} \boldsymbol{A}(\boldsymbol{\Theta})^+ \right\|_F^2.$$

This parameterization was carefully designed to simplify optimization. Because $\boldsymbol{A}(\boldsymbol{\Theta})$ is full rank, the constraints are satisfied by construction, and as a result the only constraint we need to handle is non-negativity of $\boldsymbol{\Theta}$. Furthermore, $\|\boldsymbol{A}(\boldsymbol{\Theta})\|_1 = 1$ for all $\boldsymbol{\Theta}$, so that term can be removed from the objective. Additionally, due to the special structure of $\boldsymbol{A}(\boldsymbol{\Theta})$ we can efficiently evaluate the objective and its gradient in $O(pn^2)$ time instead of $O(n^3)$ time. For example, when $n = 8192$ it requires > 6 minutes to evaluate the objective for a general strategy $\boldsymbol{A}$, while it takes only 1.5 seconds for a p-Identity strategy (with $p = \frac{n}{16}$), which is a $240\times$ improvement. Despite this imposed structure, $\boldsymbol{A}(\boldsymbol{\Theta})$ is still expressive enough to encode high-quality strategies. Moreover, p can always be tuned to balance expressivity with efficiency.

4.4. **Strategy Visualization.** An interested reader may wonder what the optimized strategies actually look like for some common workloads. In Figure 1, we plot three strategies designed for the workload of All Range queries. Figures 1a and 1b show the strategies produced by HDMM for Gaussian and Laplace noise, respectively, and Figure 1c shows a

[4]If a strategy did not, a query could be added to it without increasing sensitivity and addition of that query would result in error less than or equal to that of the original strategy.



(a) Cholesky strategy **(b)** p-Identity strategy (p=16) **(c)** Hierarchical strategy (b=16)

Figure 1. Visualization of three different strategies for answering the workload of All Range queries on a domain of size 256. Figures (a) and (b) show strategies optimized by HDMM, and Figure (c) shows a hierarchical strategy with branching factor 16, a previously state-of-the-art strategy for this workload [24,41]. Each row is a query, and cells are color coded according to their value in the strategy matrix. Figure (a) plots each query as one very thin row, while Figure (b) and (c) only plot the non-trivial queries as thicker rows. The diagonal queries are plotted separately as a single row above the main plot, but it actually represents 256 different queries.

previously state-of-the-art strategy for range queries. These visualizations provide interesting insight into the nature of the solution, and reveal why HDMM succeeds in reducing error.

In Figure 1a, the strategy is an upper triangular square matrix; this is by construction as it is obtained through a Cholesky decomposition.[5] In each row, weights are largest near the main diagonal, and quickly decrease the further away it gets. This can be observed from the smooth transition from yellow to green to blue in Figure 1a, and noting that the colors appear on a logarithmic scale.

In Figure 1b, the strategy was optimized with $p = 16$, but only 13 non-zero queries were found. Each query has varying width, ranging between about 16 and 64, and has greatest weight towards the middle of the query, with gradually decreasing weights away from the center. In most cases, the queries overlap with half of the two neighboring queries. The weights on the identity queries are approximately uniform throughout at around 0.5, with higher weights near the indices 0 and 256. There is a very natural reason why this structure works well for range queries. Summing up adjacent queries leads to a bigger query which looks approximately like a range query. It will have a long uniform center, and decaying weights on the edges. These decaying weights can be increased to match the uniform center by drawing on the answers from the identity queries. Thus, any range query can be answered by summing up a relatively small number of strategy query answers.

This same intuition was used in the derivation of the hierarchical strategy in Figure 1c. However, this strategy answers some range queries more effectively than others. It struggles

[5]There may be equally good strategies that do not have this upper triangular structure, but HDMM will always find one with this structure.

for queries that require summing up many identity queries. For example, it can answer the range $[0, 16)$ using one strategy query, but it requires summing up 16 strategy queries to answer the range $[8, 24)$.

5. Implicit representations for conjunctive query sets

The optimization methods we described in the previous section work well for small and modest domain sizes; we were able to run them on domains as large as $n = 8192$ (see Section 11 for a scalability analysis). However, these methods are fundamentally limited by the need to represent the workload and strategy explicitly in matrix form. It requires 0.5 gigabytes just to store a square matrix of size 8192 using 4 byte floats, and it is time consuming to perform nontrivial matrix operations on objects of this size. This limitation is not unique to our mechanism, but is shared by *all possible* methods that optimize explicitly represented workload matrices.

To overcome this scalability limitation, we propose *implicit query matrices*, which exploit structure in conjunctive query sets and offer a far more concise representation than materialized explicit matrices, while still being able to encode query sets containing an arbitrary collection of conjunctive queries. These representations are lossless; they save space by avoiding significant redundancy, rather than making approximations. As we will show later in this section, many important matrix operations can be performed efficiently using the implicit representation. This property of the representation will be essential for solving the strategy optimization problem efficiently on large domains.

5.1. Implicitly vectorized conjunctions.

Consider a predicate defined on a single attribute, A, where $|dom(A)| = n_A$. This predicate, ϕ_A, can be vectorized with respect to just the domain of A (and not the full domain of all attributes) similarly to Definition 5. When a predicate is formed from the conjunction of such single-attribute predicates, its vectorized form has a concise implicit representation in terms of the *outer product* between vectors.

Definition 15 (Outer product)**.** *The outer product between two vectors $\boldsymbol{q}_A$ and $\boldsymbol{q}_B$, denoted $\boldsymbol{q}_A \otimes \boldsymbol{q}_B$, is a vector indexed by pairs $t = (t_A, t_B)$ such that: $(\boldsymbol{q}_A \otimes \boldsymbol{q}_B)(t) = \boldsymbol{q}_A(t_A)\boldsymbol{q}_B(t_B)$.*

The outer product is useful for representing conjunctions in vector form.

Theorem 1 (Implicit vectorization)**.** *The vector representation of the conjunction $\phi = \phi_A \wedge \phi_B$ is $vec(\phi) = vec(\phi_A) \otimes vec(\phi_B)$.*

To see why this is true, observe that $\phi(t) = \phi_A(t_A)\phi_B(t_B)$, since multiplication and logical AND are equivalent for binary inputs, and this is the exact equation that defines the outer product for vectors. While the explicit representation of $vec(\phi)$ has size $n_A \cdot n_B$, the implicit representation, $vec(\phi_A) \otimes vec(\phi_B)$, requires storing only $vec(\phi_A)$ and $vec(\phi_B)$, which has size $n_A + n_B$.

Example 12. *Recall that the workload $\mathcal{W}_{SF1}$ consists of 4151 queries, each defined on a data vector of size* 500,480*. Since explicitly vectorized queries are the same size as the domain, the size of the explicit workload matrix is* $4151 \times 500{,}480$*, or* $8.3GB$*. Using the implicit representation, each query can be encoded using* $2 + 2 + 64 + 115 + 17 = 200$ *values, for a total of 3.3MB. For $\mathcal{W}_{SF1+}$, which consists of* 215,852 *queries on a data vector of size* 25,524,480*, the explicit workload matrix would require 22TB of storage. In contrast, the implicit vector representation would require 200MB.*

5.2. **Implicitly vectorized products.** Product workloads (as in Definition 2) can be encoded even more efficiently using the Kronecker product.

Definition 16 (Kronecker product)**.** *For two matrices $\boldsymbol{A} \in \mathbb{R}^{m_A \times n_A}$ and $\boldsymbol{B} \in \mathbb{R}^{m_B \times n_B}$, their Kronecker product is $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{m_A m_B \times n_A n_B}$, where rows are indexed by pairs $q = (q_A, q_B)$ and columns are indexed by pairs $t = (t_A, t_B)$ such that,*

$$(\boldsymbol{A} \otimes \boldsymbol{B})(q, t) = \boldsymbol{A}(q_A, t_A)\boldsymbol{B}(q_B, t_B).$$

The Kronecker product is a generalization of the outer product, where each row is an outer product between a pair of rows from $\boldsymbol{A}$ and $\boldsymbol{B}$. Thus, the same symbol $\otimes$ is used for both operations.

Theorem 2 (Implicit vectorization)**.** *Let Φ_A and Φ_B be two predicate sets defined on attributes A and B respectively. Then $vec(\Phi_A \times \Phi_B) = vec(\Phi_A) \otimes vec(\Phi_B)$.*

The proof of this claim follows immediately from Definition 2, since it contains a Cartesian product of conjunctions, and each conjunction is an outer product. We implicitly represent a product workload in matrix form by storing the factors of the Kronecker product ($\boldsymbol{A} = vec(\Phi_A)$ and $\boldsymbol{B} = vec(\Phi_B)$). This requires $m_A n_A + m_B n_B$ space, rather than $m_A m_B n_A n_B$ space, which is required by the explicit representation, and $m_A m_B (n_A + n_B)$ for the implicit representation using a list of outer products. Thus, the savings can be quite substantial. While Theorem 2 assumes the predicates are conjunctions, we show in Appendix A that disjunctions can be handled in a similar manner.

Example 13 (Prefix Identity Workload)**.** *Consider the predicate set $I_{\texttt{Sex}} \times P_{\texttt{Grade}}$, where the domain of the Grade attribute is $\{A, B, C, D, F\}$. In matrix form, this workload can be represented as the following 10×10 matrix.*

$$\boldsymbol{W} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Implicitly, this workload can be represented as the Kronecker product between a 2×2 matrix and a 5×5 matrix as follows:

$$\mathbb{W} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

5.3. **Workload encoding algorithm.** Given as input a logical workload $\mathcal{W}$ (as in Definition 3), the ImpVec algorithm produces an implicitly represented workload matrix with the following form:

$$\mathbb{W} = \begin{bmatrix} w_1\mathbb{W}_1 \\ \vdots \\ w_k\mathbb{W}_k \end{bmatrix} = \begin{bmatrix} w_1(\boldsymbol{W}_1^{(1)}\otimes & \dots & \otimes \boldsymbol{W}_d^{(1)}) \\ \vdots & \ddots & \vdots \\ w_k(\boldsymbol{W}_1^{(k)}\otimes & \dots & \otimes \boldsymbol{W}_d^{(k)}) \end{bmatrix}. \tag{5.1}$$

Here stacking sub-workloads is analogous to union and in formulas we will sometimes abuse notation and write an implicit union-of-products workload as $\mathbb{W}_{[k]} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$. We use blackboard bold font to distinguish an implicitly represented workload $\mathbb{W}$ from an explicitly represented workload $\boldsymbol{W}$.

Algorithm 1: ImpVec

Input: Workload $\mathcal{W} = \{q_1 \dots q_k\}$ and weights $w_1 \dots w_k$
Output: Implicit workload $\mathbb{W}$

1. For each product $q_i \in \mathcal{W}$: $q_i = \Phi_{A_1}^{(i)} \times \cdots \times \Phi_{A_d}^{(i)}$
2. For each $j \in [1..d]$
3. compute $\boldsymbol{W}_j^{(i)} = vec(\Phi_{A_j}^{(i)})$
4. Let $\mathbb{W}_i = \boldsymbol{W}_1^{(i)} \otimes \cdots \otimes \boldsymbol{W}_d^{(i)}$
5. **Return:** $w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$

Note that line 3 of the ImpVec algorithm is *explicit* vectorization, as in Definition 5, of a set of predicates on a single attribute.

Example 14. *Recall from Example 10 that the* 215,852 *queries of* $\mathcal{W}_{SF1+}$ *can be represented as* $k = 8032$ *products. If* $\mathcal{W}_{SF1+}$ *is represented in this factored form, the* ImpVec *algorithm returns a smaller implicit representation, reducing the 200MB (from Example 12) to 50 MB. If the workloads are presented in their manually factored format of* $k = 32$ *products, the implicit representation of* $\mathcal{W}^*_{SF1}$ *requires only 335KB, and* $\mathcal{W}^*_{SF1+}$ *only 687KB.*

5.4. **Operations on vectorized objects.** Reducing the size of the workload representation is only useful if critical computations can be performed without expanding them to their explicit representations. Standard properties of the Kronecker product [46] accelerate strategy selection and reconstruction.

The following properties allow us to perform useful operations on Kronecker products without materializing their full matrices.

Proposition 6 (Kronecker identities)**.** *Kronecker products satisfy the following identities [46]:*

Transpose: $(\boldsymbol{A} \otimes \boldsymbol{B})^T = \boldsymbol{A}^T \otimes \boldsymbol{B}^T$
Pseudo Inverse: $(\boldsymbol{A} \otimes \boldsymbol{B})^+ = \boldsymbol{A}^+ \otimes \boldsymbol{B}^+$
Associativity: $(\boldsymbol{A} \otimes \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes (\boldsymbol{B} \otimes \boldsymbol{C})$
Mixed Product: $(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{AC}) \otimes (\boldsymbol{BD})$

In addition to the standard properties of Kronecker product above, we can prove additional properties about them which are useful for our privacy mechanism.

Theorem 3 (Norm of a Kronecker product)**.** *The following matrix norms decompose over the factors of the Kronecker product* $\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$*.*

$$\|\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d\|_{\mathcal{L}} = \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{\mathcal{L}}$$

$$\|\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d\|_{\mathcal{G}} = \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{\mathcal{G}}$$

$$\|\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d\|_{F} = \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{F}$$

The proof is largely routine algabraic manipulation and is given in the appendix. The theorem statement is a special case of a more general result regarding the norms of Kronecker products [26]. We will later use these identities to efficiently evaluate TSE for workloads and strategies built with Kronecker products.

6. Optimizing conjunctive query workloads with conjunctive query strategies

We now turn our attention to optimizing implicitly-represented conjunctive query workloads. We assume the workload is a *union of Kronecker products*, and that it takes the form shown in Equation (5.1) (restated below):

$$\mathbb{W} = \begin{bmatrix} w_1\mathbb{W}_1 \\ \vdots \\ w_k\mathbb{W}_k \end{bmatrix} = \begin{bmatrix} w_1(\boldsymbol{W}_1^{(1)} \otimes & \dots & \otimes \boldsymbol{W}_d^{(1)}) \\ \vdots & \ddots & \vdots \\ w_k(\boldsymbol{W}_1^{(k)} \otimes & \dots & \otimes \boldsymbol{W}_d^{(k)}) \end{bmatrix}. \tag{5.1}$$

For notational convenience, we will often write $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$, where $\mathbb{W}_i = \boldsymbol{W}_1^{(i)} \otimes \cdots \otimes \boldsymbol{W}_d^{(i)}$ and '+' serves the role of stacking matrices. The methods described in this section will exploit the special structure of this class of workloads to scale more effectively than techniques described in Section 4.

6.1. Optimizing product workloads.

We begin by considering a special case of the workload in Equation (5.1) that is a *single Kronecker product*. For workloads of this form, we propose optimizing the subworkloads on each attribute individually and then take the Kronecker product of the optimized substrategies to form a strategy for the original workload. We optimize the subworkloads using OPT_0.

Definition 17 ($\mathsf{OPT}_\otimes$)**.** *Given a Kronecker product workload* $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$ *and an optimization oracle* OPT_0*, let* $\mathsf{OPT}_\otimes(\mathbb{W}) = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$ *where* $\boldsymbol{A}_i = \mathsf{OPT}_0(\boldsymbol{W}_i)$*.*

Above, OPT_0 may be any strategy optimization routine that consumes an explicitly represented workload and returns a strategy matrix, such as the techniques discussed in the previous section. Since $\mathsf{OPT}_\otimes$ requires solving d small subproblems rather than one large problem, it can be far more efficient than OPT_0 for this class of workloads. This decomposition of the objective function has a well-founded theoretical justifiication. Namely,

if we restrict the solution space to a (single) Kronecker product strategy of the form $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$, then the error of the workload under $\mathbb{A}$ decomposes over the factors of the Kronecker products as shown in the following theorem:

Theorem 4 (Error decomposition)**.** *Given a workload $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$ and a strategy $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$, the error is:*

$$\|\mathbb{A}\|_{\mathcal{K}}^2 \left\|\mathbb{W}\mathbb{A}^+\right\|_F^2 = \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{\mathcal{K}}^2 \left\|\boldsymbol{W}_i \boldsymbol{A}_i^+\right\|_F^2 .$$

The overall error is minimized when $\boldsymbol{A}_i$ optimizes $\boldsymbol{W}_i$ for each i, thus it makes sense to optimize each $\boldsymbol{A}_i$ separately. If we expect the optimal strategy to be a single Kronecker product, then this approach seems quite appealing. However it is possible that there exists a strategy that is *not* a single Kronecker product that offers lower error than the best Kronecker product strategy. The following theorem shows that this is not the case, and gives further justification for the above method, showing that the SVD bound also decomposes over the factors of the Kronecker product.

Theorem 5 (SVD bound decomposition)**.** *Given a workload $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$, the SVD bound is:*

$$SVDB(\mathbb{W}) = \prod_{i=1}^{d} SVDB(\boldsymbol{W}_i).$$

If there exist strategies $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_d$ that achieve the SVD bound for $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_d$ and we can find them, then we can construct a Kronecker product strategy $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$ that achieves the SVD bound for $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$. Since no other strategy can have lower error than the SVD bound, in these situations *the optimal strategy is a Kronecker product*. We prove a stronger claim in Appendix D that $\mathbb{A}$ is still optimal even if the factors $\boldsymbol{A}_i$ do not achieve the SVD Bound. This gives excellent justification for optimizing over the space of Kronecker products.

6.2. **Optimizing union-of-product workloads.** The approach just described is principled and effective when the workload is a single Kronecker product. We now turn our attention to the more general case where the workload is a union of Kronecker products. Here, the right approach is less clear. We define three approaches for optimizing implicit workloads in the form of Equation (5.1). Each approach restricts the strategy to a different region of the full strategy space for which optimization is tractable. The first computes a strategy consisting of a single product; it generalizes $\mathsf{OPT}_{\otimes}$. The second, OPT_{+}, can generate strategies consisting of unions of products. The third, $\mathsf{OPT}_{\mathsf{M}}$, generates a strategy of weighted marginals. The best approach to use will generally depend on the workload, and we will provide some practical guidance and intuition to understand the situations in which each method works best.

Single-product output strategy For weighted union of product workloads, if we restrict the optimization problem to a single product strategy, then the objective function decomposes as follows.

Theorem 6. *Given workload* $\mathbb{W} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$ *and strategy* $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$*, workload error is:*

$$\begin{aligned} \|\mathbb{A}\|_{\mathcal{K}}^2 \left\|\mathbb{W}\mathbb{A}^+\right\|_F^2 &= \|\mathbb{A}\|_{\mathcal{K}}^2 \sum_{j=1}^{k} w_j^2 \left\|\mathbb{W}_j\mathbb{A}^+\right\|_F^2 \\ &= \sum_{j=1}^{k} w_j^2 \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{\mathcal{K}}^2 \left\|\boldsymbol{W}_i^{(j)}\boldsymbol{A}_i^+\right\|_F^2. \end{aligned} \tag{6.1}$$

This leads to the following optimization problem:

Definition 18 (Generalized $\mathsf{OPT}_\otimes$)**.** *Given a workload* $\mathbb{W} = w_1\mathbb{W}_1 + \ldots + w_k\mathbb{W}_k$*, let* $\mathsf{OPT}_\otimes(\mathbb{W}) = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$ *where,*

$$(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_d) = \underset{\boldsymbol{A}_1,\ldots,\boldsymbol{A}_d}{\textit{minimize}} \quad \sum_{j=1}^{k} w_j^2 \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{\mathcal{K}}^2 \left\|\boldsymbol{W}_i^{(j)}\boldsymbol{A}_i^+\right\|_F^2.$$

When $k = 1$, the solution to the problem in Definition 18 is given in Definition 17, so we use matching notation and allow the $\mathsf{OPT}_\otimes$ operator to accept a single product or a union of products.

We can solve this problem efficiently by building on the optimization oracles designed in the previous section. In particular, suppose we have a black box optimization oracle $\mathsf{OPT}_0(\boldsymbol{W})$ that accepts an explicitly represented workload and gives back an explicitly represented strategy with low (ideally minimal) error on that workload. Then we use a block method that cyclically optimizes $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_d$ until convergence. We begin by initializing $\boldsymbol{A}_i = \boldsymbol{I}$ for all i. We then optimize one $\boldsymbol{A}_i$ at a time, fixing the other $\boldsymbol{A}_{i'}$ for $i' \neq i$ using OPT_0 on a carefully constructed surrogate workload $\hat{\boldsymbol{W}}$ (Equation (6.2)) that has the property that the error of any strategy $\boldsymbol{A}_i$ on $\hat{\boldsymbol{W}}$ is the same as the error of $\mathbb{A}$ on $\mathbb{W}$. Hence, the correct objective is being minimized.

$$\hat{\boldsymbol{W}}_i = \begin{bmatrix} c_1\boldsymbol{W}_i^{(1)} \\ \vdots \\ c_k\boldsymbol{W}_i^{(k)} \end{bmatrix} \quad c_j = w_j \prod_{i' \neq i} \|\boldsymbol{A}_{i'}\|_{\mathcal{K}} \left\|\boldsymbol{W}_{i'}^{(j)}\boldsymbol{A}_{i'}^+\right\|_F \tag{6.2}$$

The cost of running this optimization procedure is determined by the cost of computing $\hat{\boldsymbol{W}}_i^\top\hat{\boldsymbol{W}}_i$ and the cost of optimizing it, which takes $O(n_i^2(p_i + k))$ and $O(n_i^2 p_i \cdot \#\text{ITER})$ time respectively (assuming each $(\boldsymbol{W}^\top\boldsymbol{W})_i^{(j)}$ has been precomputed). As before, this method scales to arbitrarily large domains as long as the domain size of the sub-problems allows OPT_0 to be efficient.

Union-of-products output strategy For certain workloads, restricting to solutions consisting of a single product, as $\mathsf{OPT}_\otimes$ does, excludes good strategies, as shown in Example 15.

Example 15. *Consider the workload* $\mathbb{W} = \mathbb{W}_1 + \mathbb{W}_2$ *where* $\mathbb{W}_1 = \boldsymbol{P} \otimes \boldsymbol{T}$ *and* $\mathbb{W}_2 = \boldsymbol{T} \otimes \boldsymbol{P}$ *on a 2-dimensional domain of size* 100×100*. Running* $\mathsf{OPT}_\otimes$ *on this workload leads to an optimized strategy of the form* $\mathbb{A} = \boldsymbol{A}_1 \otimes \boldsymbol{A}_2$*. The expected error of this strategy is* 33385*, which is much higher than it should be for such a simple workload. The poor expected error can be explained by the fact that to support the workload, both* $\boldsymbol{A}_1$ *and* $\boldsymbol{A}_2$ *have to be full rank. This means* $\mathbb{A}$ *has to include at least* 100^2 *queries, even though* $\mathbb{W}$ *only contains* 200 *queries.*

A better alternative would be to optimize $\mathbb{W}_1$ *and* $\mathbb{W}_2$ *separately using* $\mathsf{OPT}_\otimes$*. Doing this we would end up with a strategy* $\mathbb{A} = \mathbb{A}_1 + \mathbb{A}_2$*, where* $\mathbb{A}_1$ *optimizes* $\mathbb{W}_1$ *and* $\mathbb{A}_2$ *optimizes* $\mathbb{W}_2$*. The resulting strategy is much smaller because* $\mathsf{OPT}_\otimes(\mathbb{A}_1) = \mathsf{OPT}_0(\boldsymbol{P}) \otimes \mathsf{OPT}_0(\boldsymbol{T})$*, and* $\mathsf{OPT}_0(\boldsymbol{T}) = \boldsymbol{T}$*. In fact, it only contains* 212 *queries and attains an expected error of* 14252*, which is a* 2.34× *improvement.*

Based on this example, we would like a principled approach to optimize over the space of strategies that are a union of Kronecker products. Unfortunately, computing the workload error exactly for a strategy of this form is intractable, as the pseudo inverse may not be a union of Kronecker products. This makes optimization over this space of strategies challenging. We thus propose the following heuristic optimization routine inspired by Example 15. This optimization routine individually optimizes each subworkload $\mathbb{W}_j$ using $\mathsf{OPT}_\otimes$, and then combines the strategies all together to form a single strategy. It simply requries calling $\mathsf{OPT}_\otimes$ a number of times and computing appropriate weights for each optimized strategy.

Definition 19 (OPT_+)**.** *Given a workload* $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$*, let* $\mathsf{OPT}_+(\mathbb{W}) = c_1\mathbb{A}_1 + \cdots + c_k\mathbb{A}_k$ *where* $\mathbb{A}_j = \mathsf{OPT}_\otimes(\mathbb{W}_j)$ *and*

$$c_j \propto \frac{1}{\|\mathbb{A}_j\|_{\mathcal{K}}} \begin{cases} \sqrt[3]{2E_j} & \text{if } \mathcal{K} = \mathcal{L} \\ \sqrt[4]{E_j} & \text{if } \mathcal{K} = \mathcal{G} \end{cases},$$

for $E_j = w_j^2 \|\mathbb{A}_j\|_{\mathcal{K}}^2 \left\|\mathbb{W}_j\mathbb{A}_j^+\right\|_F^2$.

Above, we assume that $\mathbb{A}_j$ will be used to answer $\mathbb{W}_j$, and c_j is the weight on $\mathbb{A}_j$: it corresponds the portion of the privacy budget that will be spent to answer those queries. It is chosen to minimize total workload error. Specifically, if we allocate c_j budget to answer $\mathbb{A}_j$, then the error will be E_j/c_j^2. Thus, the choice of c_j above is based on minimizing $\sum E_j/c_j^2$ subject to the constraint $\sum |c_j| = 1$ (for Laplace noise) and $\sum c_j^2 = 1$ (for Gaussian noise). We solve this minimization problem exactly, in closed from, using the method of Lagrange multipliers.

We remark that in the definition above, $\mathbb{W}$ is split up into k sub-workloads $\mathbb{W}_1, \ldots, \mathbb{W}_k$. Each subworkload $\mathbb{W}_j$ is assumed to be a single Kronecker product, but the optimization routine is still well defined even if $\mathbb{W}_j$ is a union of Kronecker products. This opens up a nice opportunity: to group the subworkloads into clusters which will be optimized together with $\mathsf{OPT}_\otimes$. Intuitively, if two subworkloads are similar, it may make sense to group them together to optimize collectively. We do not provide an automated way to group subworkloads in this paper. This is a hard problem in general, and is out of scope for this paper. A domain expert can work out good clusterings on a case-by-case basis, or they can settle for the default clustering (one Kronecker product per cluster).

7. Implicit representations for marginal query sets

In the previous section we described $\mathsf{OPT}_\otimes$ and OPT_+, two methods for optimizing implicitly represented conjunctive query workloads. These methods differ primarily in the space of strategies they search over. Our final optimization method, OPT_M, optimizes over the space of marginal query matrices, and offers a preferable alternative to $\mathsf{OPT}_\otimes$ and OPT_+ in some settings. In order to develop these ideas formally, we must introduce substantial

new notation to enable us to work with marginal query matrices and related objects. In this section, we propose an implicit representation for marginal query sets that is even more compact than our other representation for general conjunctive query sets. We further show that these matrices can be operated on efficiently, allowing us to solve the strategy optimization problem for large multi-dimensional domains.

A marginal query matrix is a special case of a union of Kronecker products, where each Kronecker product encodes the queries to compute a single marginal (i.e., all the factors are either Identity or Total). First note that a marginal on a d-dimensional domain can be specified by a subset of elements of $\{1, \dots, d\}$. Hence, there are a total of 2^d possible marginals, and each one can be specified by an element of the set $[2^d] = \{0, \dots, 2^d - 1\}$. The most natural correspondence between these integers and the associated marginals is based on the binary representation of the integer. The query set required to compute the a^{th} marginal would be represented by $\boldsymbol{Q}_1 \otimes \cdots \otimes \boldsymbol{Q}_d$ where $\boldsymbol{Q}_i = \boldsymbol{I}$ if the i^{th} bit of the binary representation of a is 1 and $\boldsymbol{Q}_i = \boldsymbol{T}$ otherwise. A collection of weighted marginals can thus be represented as a vector $\boldsymbol{u}$ containing a weight for each marginal. We refer to this marginal query matrix as $\mathbb{M}(\boldsymbol{u})$, which is defined below.

Definition 20 (Marginal query matrix)**.** *A marginal query matrix $\mathbb{M}(\boldsymbol{u})$ is defined by a vector of weights $\boldsymbol{u} \in \mathbb{R}^{2^d}$ and is a special case of the query matrix shown in Equation (5.1) where $k = 2^d$, $w_{a+1} = \boldsymbol{u}(a)$, and*

$$\boldsymbol{W}_i^{(a+1)} = \begin{cases} \boldsymbol{I} & a_i = 1 \\ \boldsymbol{T} & a_i = 0 \end{cases},$$

where $a \in \{0, \dots, 2^d - 1\}$,$i \in \{1, \dots, d\}$,and a_i is the i^{th} bit of the binary representation of a.

For a marginal query matrix, the weight $\boldsymbol{u}(a)$ can be interpreted as the relative *importance* of the a^{th} marginal. The example below provides further clarification on this implicit representation.

Example 16. *The workoad of all 2-way marginals on a 3-dimensional domain can be expressed as $\mathbb{M}(\boldsymbol{w})$ for $\boldsymbol{w} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$. The three non-zero entries of $\boldsymbol{w}$ appear at indices 3, 5, and 6, which in binary is 011_2, 101_2 and 110_2. Written in expanded form, this workload is:*

$$\mathbb{M}(\boldsymbol{w}) = \begin{bmatrix} 0 & (\boldsymbol{T} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{T}) \\ 0 & (\boldsymbol{T} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{I}) \\ 0 & (\boldsymbol{T} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{T}) \\ 1 & (\boldsymbol{T} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{I}) \\ 0 & (\boldsymbol{I} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{T}) \\ 1 & (\boldsymbol{I} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{I}) \\ 1 & (\boldsymbol{I} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{T}) \\ 0 & (\boldsymbol{I} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{I}) \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{T} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{I} \\ \boldsymbol{I} & \otimes & \boldsymbol{T} & \otimes & \boldsymbol{I} \\ \boldsymbol{I} & \otimes & \boldsymbol{I} & \otimes & \boldsymbol{T} \end{bmatrix}.$$

As shown in Proposition 7, it is particularly simple to reason about the sensitivity of a marginal query matrix.

Proposition 7. *The sensitivity of a marginal query matrix $\mathbb{M}(\boldsymbol{u})$ is:*

$$\|\mathbb{M}(\boldsymbol{u})\|_{\mathcal{L}} = \|\boldsymbol{u}\|_1, \qquad \|\mathbb{M}(\boldsymbol{u})\|_{\mathcal{G}} = \|\boldsymbol{u}\|_2.$$

Moving forward, it is convenient to work with the Gram matrix representation of the marginal query matrix instead. As shown below, there is a simple correspondence between the two.

Proposition 8 (Marginal Gram matrix)**.** *Let $\mathbb{Q} = \mathbb{M}(\boldsymbol{u})$ be a marginal query matrix. Then the correpsonding marginal Gram matrix is $\mathbb{Q}^\top\mathbb{Q} = \mathbb{G}(\boldsymbol{u}^2)$, where $\boldsymbol{u}^2$ is the element-wise square of $\boldsymbol{u}$, and*

$$\mathbb{G}(\boldsymbol{v}) = \sum_{a=0}^{2^d-1} \boldsymbol{v}(a)\mathbb{H}(a), \qquad \mathbb{H}(a) = \bigotimes_{i=1}^{d} [\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)].$$

In the proposition above, the term $\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)$ is shorthand notation for $\mathbf{1}$ if $a_i = 0$ and $\boldsymbol{I}$ if $a_i = 1$. Both $\mathbf{1}$ and $\boldsymbol{I}$ are $n_i \times n_i$ matrices, corresponding to the matrix of all ones, and the identity matrix respectively. We will use this notation frequently in the section, so it is important to understand the exact meaning. Another important object that will appear repeatedly throughout this section is the so-called characteristic vector[6], which is defined below.

Definition 21 (Characteristic Vector)**.** *The characteristic vector $\boldsymbol{c} \in \mathbb{R}^{2^d}$ is defined so that each entry $\boldsymbol{c}(a)$ equals the number of entries in the $(\neg a)^{th}$ marginal,*

$$\boldsymbol{c}(a) = \prod_{i=1}^{d} n_i(a_i = 0) + 1(a_i = 1).$$

The term $\neg a$ in the definition above is the bitwise negation of a, and it is obtained by flipping each of the d bits of the integer a. We will rely heavily on this type of bitwise manipulation in this section to reason about the behavior of marginal Gram matrices.

Now that we have introduced the necessary notation for marginal query and Gram matrices, we are ready to show how to perform important matrix operations while respecting the implicit representation. We begin with Theorem 7, which shows that marginal Gram matrices interact nicely under matrix multiplication.

Theorem 7 (Multiplication of Marginal Gram Matrices)**.** *For any $a, b \in [2^d]$,*

$$\mathbb{H}(a)\mathbb{H}(b) = \boldsymbol{c}(a|b)\mathbb{H}(a\&b),$$

where $a|b$ denotes "bitwise or", $a\&b$ denotes "bitwise and", and $\boldsymbol{c}$ is the characteristic vector. Moreover, for any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{2^d}$,

$$\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) = \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{v}),$$

where $\boldsymbol{X}(\boldsymbol{u})$ is a $2^d \times 2^d$ triangular matrix with entries $\boldsymbol{X}(\boldsymbol{u})(k, b) = \sum_{a:a\&b=k} \boldsymbol{u}(a)\boldsymbol{c}(a|b)$.

Theorem 7 allows us to efficiently multiply two matrices while maintaining the compact implicit representation. Additionally, it follows immediately from the proof of Theorem 7 that $\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) = \mathbb{G}(\boldsymbol{v})\mathbb{G}(\boldsymbol{u})$ — i.e., matrix multiplication is commutative. We can apply Theorem 7 to efficiently find the inverse or generalized inverse of $\mathbb{G}(\boldsymbol{u})$ as well.

Theorem 8 (Inverse of Marginal Gram Matrices)**.** *Let $\boldsymbol{X}(\boldsymbol{u})$ be the matrix defined in Theorem 7. If $\boldsymbol{X}(\boldsymbol{u})$ is invertible, then $\mathbb{G}(\boldsymbol{u})$ is invertible with inverse:*

$$\mathbb{G}^{-1}(\boldsymbol{u}) = \mathbb{G}(\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z}),$$

[6]This is not to be confused with the eigenvector.

where $\boldsymbol{z}(2^d - 1) = 1$ *and* $\boldsymbol{z}(a) = 0$ *for all other* a*. Moreover, if* $\boldsymbol{X}^g(\boldsymbol{u})$ *is a generalized inverse of* $\boldsymbol{X}(\boldsymbol{u})$*, then a generalized inverse of* $\mathbb{G}(\boldsymbol{u})$ *is given by,*

$$\mathbb{G}^g(\boldsymbol{u}) = \mathbb{G}(\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}).$$

Because $\boldsymbol{X}(\boldsymbol{u})$ is a triangular matrix, we can compute $\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z}$ efficiently in $O(4^d)$ time using back-substitution (quadratic in the size of $\boldsymbol{z}$). Note that $\mathbb{G}(\boldsymbol{u})$ and $\boldsymbol{X}(\boldsymbol{u})$ are invertible if and only if $\boldsymbol{u}(2^d - 1) > 0$. The generalized inverse result holds even for non-invertible matrices. This result is slightly more complicated, but is important because we generally expect $\mathbb{G}$ to be singular (e.g., if it is the Gram matrix of some workload of low-dimensional marginal query matrices).

As we show in Theorem 9, we know the eigenvectors and eigenvalues of marginal Gram matrices. Recall that $\boldsymbol{v}$ is an eigenvector with corresponding eigenvalue λ if $\mathbb{G}(\boldsymbol{w})\boldsymbol{v} = \lambda\boldsymbol{v}$ for some real-valued λ. We use the term *eigenmatrix* to refer to a matrix where each column is an eigenvector that shares the same eigenvalue.

Theorem 9 (Eigenvectors and Eigenvalues of Marginal Gram Matrices)**.** *Let* $a \in [2^d]$ *and let*

$$\mathbb{V}(a) = \bigotimes_{i=1}^{d} (a_i = 0)\boldsymbol{T} + (a_i = 1)(\boldsymbol{1} - n_i\boldsymbol{I}).$$

For any $b \in [2^d]$*,* $\mathbb{V}(a)$ *is an eigenmatrix of* $\mathbb{H}(b)$ *with corresponding eigenvalue* $\boldsymbol{\lambda}(a) = \boldsymbol{c}(b)$ *if* $a\&b = a$ *and* $\boldsymbol{\lambda}(a) = 0$ *otherwise. Moreover, for any* $\boldsymbol{w} \in \mathbb{R}^{2^d}$*,* $\mathbb{V}(a)$ *is an eigenmatrix of* $\mathbb{G}(\boldsymbol{w})$ *with corresponding eigenvalue* $\boldsymbol{\kappa}(a) = \sum_{b:a\&b=a} \boldsymbol{w}(b)\boldsymbol{c}(b)$*. That is,*

$$\mathbb{H}(b)\mathbb{V}(a) = \boldsymbol{\lambda}(a)\mathbb{V}(a), \qquad \mathbb{G}(\boldsymbol{w})\mathbb{V}(a) = \boldsymbol{\kappa}(a)\mathbb{V}(a).$$

Interestingly, the eigenmatrices (and hence the eigenvectors) are the same for all marginal Gram matrices $\mathbb{G}(\boldsymbol{w})$. Furthermore, the corresponding eigenvalues have a very simple (linear) dependence on the weights $\boldsymbol{w}$. In fact, there is a triangular matrix $\boldsymbol{Y}$ such that $\boldsymbol{\kappa} = \boldsymbol{Y}\boldsymbol{w}$.

8. Optimizing conjunctive query workloads with marginal query strategies

In this section, we describe $\mathsf{OPT}_{\mathsf{M}}$, an optimization operator that consumes a conjunctive query workload $\mathbb{W}$ and returns a marginal query strategy $\mathbb{A} = \mathbb{M}(\boldsymbol{\theta})$.[7] Theorem 10 is the first key to our approach for this problem. Intuitively, it states that for any conjunctive query workload $\mathbb{W}$, there is a marginal Gram matrix $\mathbb{G}(\boldsymbol{w})$ that is equivalent to $\mathbb{W}^\top\mathbb{W}$ for the purposes of optimization.

Theorem 10 (Marginal approximation of conjunctive query workload)**.** *For any conjunctive query workload* $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$*, there is a marginal Gram matrix* $\mathbb{G}(\boldsymbol{w})$[8] *such that* $tr[\mathbb{G}(\boldsymbol{u})\mathbb{W}^\top\mathbb{W}] = tr[\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{w})]$ *for all* $\boldsymbol{u}$*.*

$\mathbb{G}(\boldsymbol{u})$ in Theorem 10 represents $(\mathbb{A}^\top\mathbb{A})^+$ in the expected error formula. We know this is a marginal Gram matrix by Proposition 8 and Theorem 8. Theorem 10 allows us to reduce the problem of optimizing an arbitrary conjunctive query workload to simply optimizing a marginal query workload, which we can do efficiently. In fact, as we show in Theorem 11, we can efficiently evaluate the matrix mechanism objective for marginal query strategies, which is essential for efficient optimization.

[7] We reserve the symbol $\boldsymbol{\theta}$ for *strategies*, and use $\boldsymbol{u}, \boldsymbol{v}$ and $\boldsymbol{w}$ to refer to other marginal Gram matrices.

[8] $\boldsymbol{w}$ is related to, but not equal to $w_1, \ldots, w_k$; $\boldsymbol{w}$ has size $2^d \neq k$.

Theorem 11 (Marginal parameterization objective function)**.** *Let* $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$ *be a conjunctive query workload and let* $\mathbb{G}(\boldsymbol{w})$ *be the marginal approximation of* $\mathbb{W}^\top\mathbb{W}$ *(as in Theorem 10). For any marginal query strategy* $\mathbb{A} = \mathbb{M}(\boldsymbol{\theta})$*, the matrix mechanism objective function can be expressed as,*

$$\|\mathbb{A}\|_{\mathcal{K}}^2 \left\|\mathbb{W}\mathbb{A}^+\right\|_F^2 = \|\boldsymbol{\theta}\|_{\mathcal{K}}^2 [\mathbf{1}^\top \boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w}].$$

where $\|\boldsymbol{\theta}\|_{\mathcal{K}}$ *is the sensitivity norm defined in Proposition 7, and* $\boldsymbol{X}$ *is the matrix defined in Theorem 7.*

Theorem 11 shows that we can efficiently calculate the objective function in terms of $\boldsymbol{w}$ and $\boldsymbol{\theta}$, without ever explicitly materializing $\mathbb{G}(\boldsymbol{w})$ or $\mathbb{M}(\boldsymbol{\theta})$. This key idea will allow us to solve the strategy selection problem efficiently. Problem 2 states the main optimization problem that underlies $\mathsf{OPT}_{\mathsf{M}}$, which immediately follows from Theorem 11.

Problem 2 (Marginals parameterization)**.** *Given a conjunctive query workload* $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$*, let* $\mathsf{OPT}_{\mathsf{M}}(\mathbb{W}) = \mathbb{M}(\boldsymbol{\theta}^*)$ *where*

$$\begin{aligned} \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \quad & \|\boldsymbol{\theta}\|_{\mathcal{K}}^2 [\mathbf{1}^\top \boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w}] \\ \textit{subject to} \quad & \boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}(\boldsymbol{\theta}^2)\boldsymbol{w} = \boldsymbol{w} \end{aligned}$$

and $\mathbb{G}(\boldsymbol{w})$ *is the marginal approximation of* $\mathbb{W}^\top\mathbb{W}$ *(as in Theorem 10).*

Above, the constraint ensures that the strategy supports the workload. In practice, this constraint can usually be ignored, and the resulting unconstrained optimization problem can be solved instead. The constraint can then be verified to hold at the end of the optimization. Intuitively, this is because strategies that move closer to the boundary of the constraint will have higher error, so the optimization will never approach it as long as sufficiently small step sizes are taken. We use scipy.optimize to solve this problem in practice.

We note that the number of parameters in the above optimization problem is 2^d and that we can evaluate the objective in $O(4^d)$ time (quadratic in the number of parameters). Thus, it is feasible to solve this problem as long as $d \leq 15$. Importantly, this means that the runtime complexity is independent of the domain size of each attribute, so it will take the same amount of time for $n_i = 2$ (binary features), $n_i = 10$, or any other values of n_i.

In addition to being able to efficiently optimize over the space of marginal query strategies, we can also efficiently compute the SVD bound for marginal query workloads. Theorem 12 gives a remarkably simple formula for computing the SVD bound for marginal query workloads.

Theorem 12 (SVD Bound for Marginal Query Workloads)**.** *The SVD bound for a marginal query workload* $\mathbb{W}$ *with Gram matrix* $\mathbb{G}(\boldsymbol{w})$ *is,*

$$SVDB(\mathbb{W}) = \frac{1}{n}\Big(\sum_a \boldsymbol{c}(\neg a)\sqrt{\sum_{b:a\&b=a} \boldsymbol{w}(b)\boldsymbol{c}(b)}\Big)^2.$$

Additionally, as a byproduct of this analysis, we give a similarly simple formula to *find the optimal marginal query strategy in closed form* in Theorem 13, allowing us to bypass the need for numerical optimization in some settings.

Theorem 13 (Closed form solution to Problem 2)**.** *Let $\mathbb{W}$ be a workload with Gram matrix $\mathbb{G}(\boldsymbol{w})$ and let $\boldsymbol{\theta} = \sqrt{\boldsymbol{Y}^{-1}\sqrt{\boldsymbol{Y}\boldsymbol{w}}}$ (element-wise square root), where $\boldsymbol{Y}$ is the $2^d \times 2^d$ matrix:*

$$\boldsymbol{Y}(a,b) = \begin{cases} \boldsymbol{c}(b) & a\&b = a \\ 0 & otherwise \end{cases}.$$

If $\boldsymbol{\theta}$ contains real-valued entries then the strategy $\mathbb{A} = \mathbb{M}(\boldsymbol{\theta})$ attains the SVDB bound when $\mathcal{K} = \mathcal{G}$, and is thus an optimal strategy. That is, $\|\mathbb{A}\|_{\mathcal{G}}^2 \|\mathbb{W}\mathbb{A}^+\|_F^2 = SVDB(\mathbb{W})$.

While $\boldsymbol{\theta}$ may sometimes contain imaginary entries, we can always fall back on numerical optimization to solve Problem 2. The formula in Theorem 13 can still be used to initialize the optimization if the imaginary entries of $\boldsymbol{\theta}$ are replaced with zeros. Li and Miklau derived sufficient conditions for the SVD bound to be realizable [30], and marginal query workloads satisfy those sufficient conditions. This implies that the SVD bound should always be attainable for workloads of this form. If the parameters in Theorem 13 contain imaginary entries, this suggests that the optimal strategy that is *not* a marginal query strategy. It is an interesting open question to determine what the structure of the optimal strategy is when Theorem 13 does not apply. In practice, even when the SVD bound is not attained exactly by $\mathsf{OPT}_{\mathsf{M}}$, we get very close to it for marginal query workloads, as we show empirically in Table 7 of the experiments.

9. The $\mathsf{OPT}_{\mathsf{HDMM}}$ strategy selection algorithm

	Definition	Operator	Input workload	Output strategy	Complexity
§4	Definitions 12 and 14	OPT_0	Explicit matrix $\boldsymbol{W}$	Explicit matrix $\boldsymbol{A}$	$O(n^3)$
§6.1	Definition 17	$\mathsf{OPT}_\otimes$	Kronecker Product	Kronecker Product	$O(\sum_{i=1}^d n_i^3)$
§6.2	Definition 18	$\mathsf{OPT}_\otimes$	Union of Kronecker Products	Kronecker Product	$O(k\sum_{i=1}^d n_i^3)$
§6.2	Definition 19	OPT_+	Union of Kronecker Products	Union of Kronecker Products	$O(k\sum_{i=1}^d n_i^3)$
§8	Problem 2	$\mathsf{OPT}_{\mathsf{M}}$	Union of Kronecker Products	Marginal Query Strategy	$O(4^d)$

Table 2. Summary of optimization operators: input and output types, and the time complexity of objective/gradient calculations. $n = n_1 \times \cdots \times n_d$ refers to the domain size, and k (where applicable) refers to the number of Kronecker products in the workload.

In this paper, we proposed four optimization routines: OPT_0, $\mathsf{OPT}_\otimes$, OPT_+, and $\mathsf{OPT}_{\mathsf{M}}$. In this section, we summarize these different approaches, discuss the pros and cons of each one, and propose a meta-optimization algorithm $\mathsf{OPT}_{\mathsf{HDMM}}$ that automatically chooses the best one based on the workload. Table 2 summarizes the basic inputs and outputs of each operator. OPT_0 is designed to optimize an explicitly represented workload, and returns an explicitly represented strategy. The other optimization operators all operate in an implicit space however.

The time complexity of OPT_0 is $O(n^3)$ (where n is the domain size), and it generally feasible to run as long as $n \leq 10^4$. The time complexity of $\mathsf{OPT}_\otimes$ and OPT_+ is $O(k\sum n_i^3)$, where k is the number of union terms in the workload, and n_i is the domain size of attribute i. It is generally feasible to run as long as OPT_0 is feasible on each of the individual attributes (i.e., $n_i \leq 10^4$). In contrast to OPT_0, the total domain size for these operators can be arbitrarily large. The time complexity of $\mathsf{OPT}_{\mathsf{M}}$ is $O(4^d)$, which interestingly does not

depend on the domain size of individual attributes, only the number of attributes. It is generally feasible to run as long as $d \leq 15$.

Each of the operators searches over a different space of strategies, and the best one to use will ultimately depend on the workload. We illustrate the behavior of each optimization operator on the simple workload of all 2-way marginals in Example 17. This example highlights and summarizes the key differences between $\mathsf{OPT}_\otimes$, OPT_+, and OPT_M. In this case, OPT_M is the best, which is not surprising because it is the most suitable for marginal workloads. It achieves this by placing more weight on the queries for larger marginals, and less weight on other queries. When compared to the baseline of using $\mathbb{W}$ as the strategy, OPT_M achieves lower error on the larger marginals but has higher error on the smaller marginals. As a result, OPT_M enjoys lower *overall* error than the simple baseline, but suffers higher *max* error. The expected errors reported in Example 17 pertain to TSE from Definition 9.

In general, predicting which optimization operator will yield the lowest error strategy requires domain expertise and may be challenging for complex workloads. Since strategy selection is independent of the input data and does not consume the privacy budget, we can just run each optimization operator, keeping the output strategy that offers the smallest expected error. Additionally, since the strategies found by each optimization operator may depend on the initialization, we recommend running several random restarts of each optimization operator, returning the best one.

By default, $\mathsf{OPT}_\mathsf{HDMM}$ invokes all three high-dimensional optimization operators $\mathsf{OPT}_\otimes$, OPT_+, and OPT_M. (OPT_0 may also be included for lower-dimensional workloads.) For $\mathsf{OPT}_\otimes$ and OPT_+ invoked with the p-Identity strategy we use the following convention for setting the p parameters: if an attribute's subworkload is completely defined in terms of $\boldsymbol{T}$ and $\boldsymbol{I}$, we set $p = 1$ (this is a fairly common case where more expressive strategies do not help), otherwise we set $p = n_i/16$ for each attribute A_i with size n_i.

Example 17 (Optimizing Marginal Query Workload)**.** *Consider the workload containing queries to compute all 2-way marginals on a domain of size* $(2, 5, 50, 100)$*. This workload can be represented as a union of* $\binom{4}{2} = 6$ *Kronecker products. Table 3 gives the precise representation of this workload, together with the optimized strategies found by* $\mathsf{OPT}_\otimes$*,* OPT_+*, and* OPT_M*. All optimized strategies can be expressed in terms of the "Identity" (*$\boldsymbol{I}$*) and "Total" (*$\boldsymbol{T}$*) building blocks.* $\mathsf{OPT}_\otimes$ *tries to find the best single Kronecker product strategy, while* OPT_+ *tries to find the optimal weight to assign to each of the six Kronecker products that make up the workload.* OPT_M *identifies a different set of marginal queries from which all 2-way marginals can be derived. Among the three optimization operators,* OPT_M *is the best, followed by* OPT_+ *and then* $\mathsf{OPT}_\otimes$*.* OPT_M *offers a* $4.8\times$ *reduction in Expected TSE over the Identity baseline, and a* $3.3\times$ *reduction over the Workload baseline.*

	Query Matrix	Expected TSE
$\mathbb{W}$	$\begin{matrix} \boldsymbol{T}\otimes\boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{I} \\ \boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{I} \\ \boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{T} \\ \boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{T}\otimes\boldsymbol{I} \\ \boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{T} \\ \boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{T} \end{matrix}$	$206,964$
$\mathbb{I}$	$\boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{I}$	$300,000$
$\mathsf{OPT}_\otimes(\mathbb{W})$	$\boldsymbol{I}\otimes\boldsymbol{I}\otimes\begin{bmatrix}0.80\,\boldsymbol{I}\\0.20\,\boldsymbol{T}\end{bmatrix}\otimes\begin{bmatrix}0.82\,\boldsymbol{I}\\0.18\,\boldsymbol{T}\end{bmatrix}$	$213,270$
$\mathsf{OPT}_+(\mathbb{W})$	$\begin{matrix} 0.39\ \boldsymbol{T}\otimes\boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{I} \\ 0.18\ \boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{I} \\ 0.14\ \boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{T} \\ 0.13\ \boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{T}\otimes\boldsymbol{I} \\ 0.11\ \boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{T} \\ 0.05\ \boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{T} \end{matrix}$	$85,070$
$\mathsf{OPT}_\mathsf{M}(\mathbb{W})$	$\begin{matrix} 0.44\ \boldsymbol{T}\otimes\boldsymbol{T}\otimes\boldsymbol{I}\otimes\boldsymbol{I} \\ 0.31\ \boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{T}\otimes\boldsymbol{I} \\ 0.25\ \boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{T} \end{matrix}$	62,886

Table 3. A workload containing queries to compute *all 2-way marginals* on a four-dimensional domain of size $(2, 5, 50, 100)$. The optimal strategy found by each parameterization, and the respective error, is also shown. The Identity and Total query matrices $\boldsymbol{I}$ and $\boldsymbol{T}$ are color coded for readability.

10. Efficient MEASURE and RECONSTRUCT

Now that we have fully described how HDMM solves the strategy selection problem, we are ready to discuss how to run the remainder of the mechanism. Recall from Definition 10 that the matrix mechanism is defined as $\mathcal{M}_{\boldsymbol{A},\mathcal{K}}(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{W}\boldsymbol{A}^{+}\mathcal{K}(\boldsymbol{A}, \boldsymbol{x})$. With explicitly represented matrices, this computation can be done directly without problem. However, HDMM replaces the explicitly represented matrices $\boldsymbol{W}$ and $\boldsymbol{A}$ with implicitly represented ones $\mathbb{W}$ and $\mathbb{A}$, and it is no longer obvious how to run the mechanism. Conceptually, these steps can be broken down as follows. In the MEASURE step, we have to compute the noisy strategy query answers, $\boldsymbol{y} = \mathbb{A}\boldsymbol{x} + \boldsymbol{\xi}$. In the RECONSTRUCT step, we have to estimate the data vector and workload query answers, i.e., compute $\hat{\boldsymbol{x}} = \mathbb{A}^{+}\boldsymbol{y}$ and return $\mathbb{W}\hat{\boldsymbol{x}}$. A necessary key subroutine to solve these problems is to compute matrix-vector products where the matrix is a Kronecker product. Importantly, we must do this without ever materializing $\mathbb{A}$ explicitly, as that is infeasible for large domains.

Theorem 14 (Efficient matrix-vector multiplication)**.** *Let $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$ and let $\boldsymbol{x}$ be a data vector of compatible shape. Then Algorithm 2 computes the matrix-vector product $\mathbb{A}\boldsymbol{x}$. Furthermore, if $\boldsymbol{A}_i \in \mathbb{R}^{n_i \times n_i}$ and $n = \prod n_i$ is the size of $\boldsymbol{x}$ then Algorithm 2 runs in $O(n \sum n_i)$ time.*

Algorithm 2 is correct even if the factors of $\mathbb{A}$ are not square, although the time complexity is not as clean when written down.

Algorithm 2 Kronecker Matrix-Vector Product

```
procedure KMATVEC(A_1, ..., A_d, x)
    m_i, n_i = SHAPE(A_i)
    r = n
    f_{d+1} = x
    for i = d, ..., 1 do
        Z = RESHAPE(f_{i+1}, n_i, r/n_i)
        r = r · m_i/n_i
        f_i = RESHAPE(A_i Z, r, 1)
    end for
    return f_1
end procedure
```

Since all of the strategies found by our optimization routines are either Kronecker products or unions of Kronecker products, we can directly apply Algorithm 2 to efficiently implement the MEASURE step of HDMM. Note that computing the matrix-vector product for a union of Kronecker products is a trivial extension of Algorithm 2: it simply requires calling Algorithm 2 for each Kronecker product and concatenating the results into a single vector.

We can also use Algorithm 2 to efficiently implement the RECONSTRUCT step of HDMM. The main challenge is to compute $\mathbb{A}^{+}\boldsymbol{y}$, or a pseudoinverse of $\mathbb{A}$ together with a matrix-vector product. This is done slightly differently for each type of strategy:

(1) $\mathbb{A} = \mathsf{OPT}_{\otimes}(\mathbb{W}) = \boldsymbol{A}_1 \otimes \cdots \times \boldsymbol{A}_d$. From Proposition 6 we know that $\mathbb{A}^{+} = \boldsymbol{A}_1^{+} \otimes \cdots \otimes \boldsymbol{A}_d^{+}$. That is, the pseudoinverse of a Kronecker product is still a Kronecker product. Thus, we can compute $\hat{\boldsymbol{x}} = \mathbb{A}^{+}\boldsymbol{y}$ efficiently using Algorithm 2.

(2) $\mathbb{A} = \mathsf{OPT}_{\mathsf{M}}(\mathbb{W}) = \mathbb{M}(\boldsymbol{\theta})$. From basic linear algebra, we know that $\boldsymbol{A}^{+} = (\boldsymbol{A}^{\top}\boldsymbol{A})^{+}\boldsymbol{A}^{\top}$ for any matrix $\boldsymbol{A}$. Applied to this setting, we have $\mathbb{M}^{+}(\boldsymbol{\theta}) = \mathbb{G}^{+}(\boldsymbol{\theta}^2)\mathbb{M}^{\top}(\boldsymbol{\theta})$, since we

know $\mathbb{M}^\top\mathbb{M}(\boldsymbol{\theta}) = \mathbb{G}(\boldsymbol{\theta}^2)$ by Proposition 8. From Theorem 8 we know how to compute $\mathbb{G}^+(\boldsymbol{\theta}^2)$ efficiently, and we know that it equals $\mathbb{G}(\boldsymbol{\eta})$ for some $\boldsymbol{\eta}$. We aim to compute $\mathbb{M}^+(\boldsymbol{\theta})\boldsymbol{y} = \mathbb{G}^+(\boldsymbol{\theta}^2)\mathbb{M}^\top(\boldsymbol{\theta})\boldsymbol{y}$. We can easily compute $\boldsymbol{v} = \mathbb{M}^\top(\boldsymbol{\theta})\boldsymbol{y}$ using a sequence of calls to Algorithm 2 by observing that $\mathbb{M}^\top(\boldsymbol{\theta})$ is a just several Kronecker products *horizontally* stacked together. In a similar fashion, we can compute $\hat{\boldsymbol{x}} = \mathbb{G}^+(\boldsymbol{\theta}^2)\boldsymbol{v}$ because $\mathbb{G}^+(\boldsymbol{\theta}^2)$ is just the sum of a bunch of Kronecker products, which we can handle efficiently with repeated calls to Algorithm 2.

(3) $\mathbb{A} = \mathsf{OPT}_+(\mathbb{W}) = c_1\mathbb{A}_1 + \cdots + c_k\mathbb{A}_k$. Unfortunately, for a strategy of this form, we do not have a way to efficiently compute $\mathbb{A}^+\boldsymbol{y}$. While $\mathbb{A}$ is a union of Kronecker products, the pseudoinverse is not necessarily, and we are not aware of a simple formula for the pseudoinverse of $\mathbb{A}$ at all. However, we can still produce an unbiased estimate of $\mathbb{W}\boldsymbol{x}$ by using *local least squares*. To do this, we will compute $\mathbb{W}_j\mathbb{A}_j^+\boldsymbol{y}_j$ for each $j = 1, \ldots, k$, where $\boldsymbol{y}_j$ is the answers produced for sub-strategy $\mathbb{A}_j^+$. Since $\mathbb{W}_j$ and $\mathbb{A}_j^+$ are both assumed to be Kronecker products, this can be easily achieved using Algorithm 2. While $\mathbb{W}_j\mathbb{A}_j^+\boldsymbol{y}_j$ is an unbiased estimator for $\mathbb{W}_j\boldsymbol{x}$, the main drawback is that the workload query answers will not necessarily be consistent between different j.

11. Experimental Evaluation

In this section we evaluate the accuracy and scalability of HDMM. We perform a comprehensive comparison of HDMM with a variety of other mechanisms on low and high-dimensional workloads, showing that it consistently offers lower error than competitors and works in a broader range of settings than other algorithms. We also evaluate the scalability of key components of HDMM, showing that it is capable of scaling effectively to high-dimensional settings.

In accuracy experiments, we report the Root Mean Squared Error (RMSE), which is defined as $RMSE = \sqrt{\frac{1}{m}\text{TSE}(\boldsymbol{W}, \mathcal{K})}$ for an algorithm $\mathcal{K}$. We compute this value analytically using the formulas from Proposition 4 whenever possible. We separately report results for pure differential privacy with Laplace noise and approximate differential privacy with Gaussian noise. We use $\epsilon = 1.0$ and $\delta = 10^{-6}$ in all experiments, but note that the ratio of errors between two data-independent algorithms remains the same for all values of ϵ and δ.[9]

These experiments are meant to demonstrate that HDMM offers the best accuracy in the *data-independent* regime. It is possible that some data-dependent mechanisms will outperform even the best data-independent mechanism, and this will typically depend on the amount of data available and the privacy budget [23, 47]. Data-independent mechanisms (like HDMM) are generally preferable when there is an abundance of data and/or the privacy budget is not too small, such as the U.S. Census decennial data release [1].

11.1. Evaluating OPT_0 on Low Dimensional Workloads.

We begin by studying the effectiveness of OPT_0 in the one-dimensional setting. Specifically, we evaluate the quality of the strategies found by our optimization oracle compared with other *data-independent*

[9]The ratio of errors between an $(\epsilon, 0)$-DP mechanism and an (ϵ, δ)-DP mechanism is a data-independent quantity. It will in general depend on δ, however.

ϵ-differential privacy (Laplace noise)

workload	domain	Identity	H2	Privelet	HB	GreedyH	LRM	OPT_0	SVDB
all-range	**64**	6.63	11.28	10.11	6.63	6.34	7.02	**5.55**	3.22
	256	13.11	16.27	14.87	8.90	9.72	15.73	**8.07**	4.07
	1024	26.15	21.83	20.26	12.82	14.70	-	**11.08**	4.94
	4096	52.27	27.90	26.18	16.19	22.21	-	**14.38**	5.82
prefix	**64**	8.06	9.42	9.37	8.06	6.04	7.67	**5.32**	2.89
	256	16.03	13.16	13.09	8.97	9.13	12.64	**7.35**	3.50
	1024	32.02	17.29	17.20	12.87	14.32	15.43	**9.58**	4.11
	4096	64.01	21.77	21.67	14.91	22.40	-	**12.20**	4.74
width32	**64**	8.00	12.02	11.09	8.00	7.32	9.44	**5.88**	2.75
	256	8.00	15.50	13.57	7.41	8.00	25.81	**6.34**	3.26
	1024	8.00	18.98	16.56	9.50	8.00	16.98	**6.41**	3.36
	4096	8.00	22.45	19.58	10.96	8.00	-	**6.46**	3.38
permuted	**64**	6.63	25.02	18.97	6.63	6.83	7.02	**5.55**	3.22
	256	13.11	66.25	49.09	18.48	13.02	15.73	**8.06**	4.07
	1024	26.15	157.50	117.06	37.07	23.94	-	**11.08**	4.94
	4096	52.27	374.29	277.42	107.83	45.77	-	**14.37**	5.82

Table 4. Error of strategies for 1D workloads with $\epsilon = 1.0$.

(ϵ, δ)-differential privacy (Gaussian noise)

workload	domain	Identity	H2	Privelet	HB	GreedyH	COA	OPT_0	SVDB
All Range	**64**	19.82	12.74	11.42	19.82	14.64	**9.73**	**9.73**	9.62
	256	39.18	16.20	14.81	18.80	23.34	**12.26**	**12.26**	12.15
	1024	78.13	19.66	18.24	27.07	36.20	14.89	**14.85**	14.75
	4096	156.14	23.12	21.69	27.92	56.21	17.92	**17.46**	17.38
prefix	**64**	24.08	10.64	10.58	24.08	14.04	**8.87**	**8.87**	8.62
	256	47.89	13.11	13.03	18.95	22.11	10.70	**10.66**	10.44
	1024	95.64	15.57	15.49	27.18	35.59	16.29	**12.49**	12.29
	4096	191.21	18.03	17.95	25.72	56.70	26.50	**14.32**	14.15
width32	**64**	23.90	13.57	12.52	23.90	17.30	8.79	**8.74**	8.23
	256	23.90	15.44	13.52	15.65	23.90	12.24	**9.93**	9.73
	1024	23.90	17.10	14.92	20.08	23.90	16.00	**10.08**	10.02
	4096	23.90	18.60	16.22	18.90	23.90	18.38	**10.11**	10.09
permuted	**64**	19.82	28.26	21.42	19.82	16.13	**9.73**	**9.73**	9.62
	256	39.18	65.97	48.88	39.04	35.22	**12.26**	**12.26**	12.15
	1024	78.13	141.86	105.44	78.30	60.60	14.89	**14.85**	14.75
	4096	156.14	310.11	229.85	185.98	118.03	17.92	**17.45**	17.38

Table 5. Error of strategies for 1D workloads with $\epsilon = 1.0$ and $\delta = 10^{-6}$.

mechanisms designed for this setting. It is important to understand the accuracy in the one-dimensional setting well, because OPT_0 is used as a sub-routine for the higher-dimensional optimization operators $\mathsf{OPT}_\otimes$ and OPT_+.

Workloads. We consider four different workloads: **All Range**, **Prefix**, **Width 32 Range**, and **Permuted Range**, each defined over domain sizes ranging from 64 to 4096. **All Range** contains every possible range query over the specified domain; **Prefix** contains range queries defining an empirical CDF; **Width 32 Range** contains all range queries of width 32. While the first three workloads are subsets of range queries, the last workload, **Permuted Range**,

is the result of right-multiplying the workload of all range queries by a random permutation matrix. Many proposed strategies have targeted workloads of range queries and tend to work fairly well on subsets of range queries. **Permuted Range** poses a challenge because the structure of the workload is hidden by the permutation, requiring a truly adaptive method to find a good strategy.

Note the large size of some of these workloads: **All Range** and **Permuted Range** have $\frac{n(n+1)}{2}$ queries. For large n it is infeasible to write down $\boldsymbol{W}$ in matrix form, but we can still compute the expected error since it only depends on the workload through its Gram matrix, $\boldsymbol{W}^\top \boldsymbol{W}$, which is $n \times n$ and has special structure, allowing it to be computed directly without materializing $\boldsymbol{W}$.

Mechanisms. We consider 8 competing mechanisms: **Identity**[10], **Laplace**, **Gaussian**, **LRM** [54], **COA** [53], **H2** [24], **HB** [41], **Privelet** [48], and **GreedyH** [27]. The first five mechanisms are general purpose mechanisms, designed to support virtually any workload. The last four mechanisms were specifically designed to offer low error on range query workloads. We also report **SVDB** to understand the gap between the error of the computed strategies and the best lower bound on error we have (via the SVD bound).

Results and Findings. Table 4 and Table 5 report the error of various mechanisms in each setting, for both Laplace and Gaussian noise respectively. We remind the reader that these values do not depend on the true data $\boldsymbol{x}$, and thus they hold for all $\boldsymbol{x}$. We report numbers for fixed $\epsilon = 1.0$ and $\delta = 10^{-6}$, but we note that these privacy parameters only impact the error by a constant factor, and hence the relationship between the errors of every pair of mechanisms remains the same for all (ϵ, δ). We have four main findings from these results, enumerated below:

(1) OPT_0 offers lower error than all competitors in all settings, and the magnitude of the improvement offered by HDMM (over the next best competitor) is as large as 3.18 for Laplace noise (on Permuted Range) and 1.61 for Gaussian noise (on Width 32 Range). Interestingly, OPT_0 offers lower error than **H2**, **HB**, **Privelet**, and **GreedyH** on range query workloads, even though these four mechanisms were designed specifically for range queries. In addition, the second best method after OPT_0 differs in each setting, which shows that some competing algorithms have specialized capabilities that allow them to perform well in some settings, while HDMM performs well in a variety of settings as it does not make strict assumptions about the workload.

(2) OPT_0 gets within a factor of 2.57 of the SVD bound for Laplace noise and 1.01 of the SVD bound for Gaussian noise on every tested workload. The gap between OPT_0 and **SVDB** is quite small for Gaussian noise, suggesting that OPT_0 is finding the best possible strategy. Note that **COA** also finds a optimal strategy in many of the settings, but it fails on the Prefix and Width 32 Range workloads when $n \geq 1024$, indicating non-convergence. Thus, even though it is solving the same problem underlying OPT_0 in theory, the implementation is not as robust as ours. The gap between OPT_0 and **SVDB** is larger for Laplace noise, however, and it is unclear if this gap is primarily due to looseness of the SVD bound or suboptimality of the strategy. Nevertheless, even with Laplace noise the ratio between OPT_0 and **SVDB** is at most 2.57.

[10]As the name implies, this mechanism instantiates the matrix mechanism using the identity matrix as the strategy, i.e., $\boldsymbol{A} = \boldsymbol{I}$.

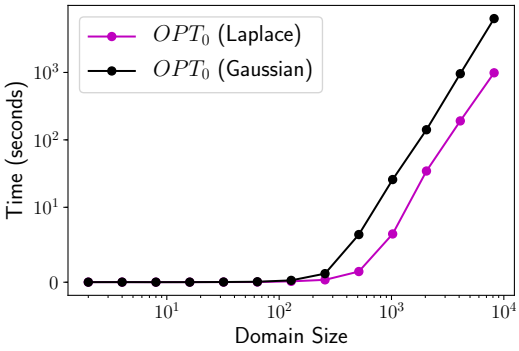


Figure 2. Time required to run OPT_0 for 100 iterations on the AllRange workload for increasing domain sizes.

(3) The error of OPT_0 (and **COA** for (ϵ, δ)-DP) is the same on the All Range and Permuted Range workloads. Permuting the workload doesn't impact achievable error or our optimization algorithm in any meaningful way. However, many of the methods we compared against perform well on All Range but poorly on Permuted Range because they were specifically designed for range queries. This shows that they exploit specific structure of the input workload and have limited adaptivity, while OPT_0 is not overly specialized to range query workloads.

(4) On these workloads, Laplace noise offers better error than Gaussian noise (for appropriately conservative settings of δ). This is because with Gaussian noise there is an additional $\approx \sqrt{\log(1/\delta)}$ term in the standard deviation of the noise, and this outweighs the benefit using the L_2 sensitivity norm instead of the L_1 sensitivity norm, despite the fact that we may be finding strategies that are further from optimal in the L_1 case.

Scalability. We now demonstrate the scalability of OPT_0. Note that optimization time dominates in the low-dimensional setting, and the time for `MEASURE` and `RECONSTRUCT` is small in comparison to that. The per-iteration time complexity only depends on the domain size, and not the contents of the workload. While the number of iterations required for convergence may differ slightly based on the queries in the workload, for simplicity we measure the time required to run the optimization for 100 iterations on the All Range workload.

Figure 2 shows the amount of time required to run OPT_0 for various domain sizes. It shows that OPT_0 scales up to $n = 8192$, and runs for $n = 1024$ in under 10 seconds for Laplace noise and 1 minute for Gaussian noise. This difference occurs because the per-iteration time complexity is $O(pn^2)$ under Laplace noise but $O(n^3)$ under Gaussian noise. For $n = 8192$ it takes considerably longer, but is still feasible to run. We remark that trading a few hours of computation time for a meaningful reduction in error is typically a welcome trade-off in practice, especially since workloads can be optimized once and the resulting strategies reused many times. Additionally, we have a prototype implementation that uses GPUs and PyTorch, and we found that it is possible (although very time consuming) to scale up to $n = 16384$. Beyond this point, it quickly becomes infeasible to even represent the workload (or its Gram matrix) in matrix form, let alone optimize it.

ϵ-differential privacy (Laplace noise)									
dataset	**workload**	**Identity**	**Laplace**	**DataCube**	**OPT$_\otimes$**	**OPT$_+$**	**OPT$_M$**	**HDMM**	**SVDB**
Census (5D)	**SF1**	23.20	70.71	-	7.30	30.55	9.56	7.30	-
	SF1+	32.50	141.42	-	10.23	42.31	12.88	10.23	-
CPS (5D)	**All Marginals**	5.38	45.25	18.49	4.85	4.85	4.84	4.84	2.63
	All Prefix-Marginals	98.06	56568.54	-	40.59	40.59	69.38	40.59	9.32
Adult (14D)	**$\leq$ 3D Marginals**	5352117.26	664.68	494.06	872.58	306.33	225.35	225.35	15.08
	2D Prefix-Marginals	475602516.60	138602.83	-	1119.16	484.07	553.56	484.07	-
Loans (12D)	**Small Marginals**	3330650.46	265.87	113.98	654.35	204.17	100.92	100.92	11.61
	Small Prefix-Marginals	15340082.96	11013.90	-	1707.67	485.67	288.29	288.29	-

Table 6. RMSE of HDMM strategies and baseline strategies on multi-dimensional workloads (ranging from 5D to 14D) for $\epsilon = 1.0$ with Laplace noise.

(ϵ, δ)-differential privacy (Gaussian noise)									
dataset	**workload**	**Identity**	**Gaussian**	**DataCube**	**OPT$_\otimes$**	**OPT$_+$**	**OPT$_M$**	**HDMM**	**SVDB**
Census (5D)	**SF1**	69.33	29.87	-	9.80	75.66	14.31	9.80	-
	SF1+	97.08	42.25	-	10.90	84.16	15.88	10.90	-
CPS (5D)	**All Marginals**	16.08	23.90	19.53	7.85	7.85	7.86	7.85	7.85
	All Prefix-Marginals	292.93	844.94	-	29.48	29.49	104.09	29.48	27.85
Adult (14D)	**$\leq$ 3D Marginals**	15988375.02	91.59	77.36	82.42	899.04	46.44	46.44	45.06
	2D Prefix-Marginals	1420766966.19	1322.58	-	126.17	639.43	296.12	126.17	-
Loans (12D)	**Small Marginals**	9949649.11	57.92	37.37	81.51	631.40	34.91	34.91	34.67
	Small Prefix-Marginals	45825415.90	372.83	-	132.43	994.04	99.72	99.72	-

Table 7. RMSE of HDMM strategies and baseline strategies on multi-dimensional workloads (ranging from 5D to 14D) for $\epsilon = 1.0$ and $\delta = 10^{-6}$ with Gaussian noise.

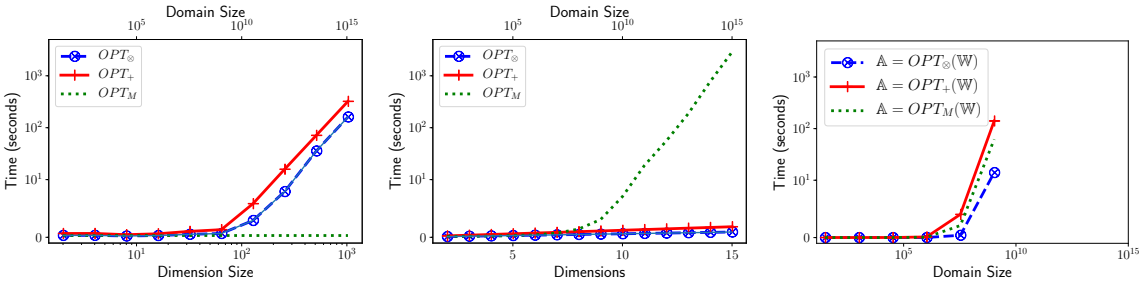
11.2. **Evaluating OPT$_\otimes$, OPT$_+$, and OPT$_M$ on Multi-Dimensional Workloads.** We now shift our attention to the multi-dimensional setting.

Workloads. We consider four multi-dimensional schemas and two workloads for each schema. The first schema, Census of Population and Housing (**Census**), has been used as a running example throughout the paper. The second schema, Current Population Survey (**CPS**), is another Census product. These schemas have 5 attributes each and domain sizes of about 1 million. The last two schemas, **Adult** and **Loans** are much higher-dimensional, having 15 and 12 attributes respectively.

For the **Census** schema, we use the SF1 and SF1+ workloads introduced in the paper. For the other schemas, we use workloads based on Marginals and Prefix-Marginals, as defined in Example 8. For **CPS** we use the workload of All Marginals and All Prefix-Marginals. For Adult, we use All $0, 1, 2$, and 3-way marginals and all 2-way Prefix-Marginals. For Loans, we use All Small Marginals and All Small Prefix-Marginals. A "Small" Marginal can be any k-way Marginal with less than 5000 cells. This means the workload will be an interesting combination of $0, 1, 2, \ldots, k$-way marginals.

We note that for the Adult and Loans schema, the domain is far too large to allow $\boldsymbol{x}$ to be represented in vector form. As a result, running HDMM as described in this paper would not be feasible. However, we remind the reader that in this section we are simply reporting *expected errors*, which we can compute efficiently without ever materializing $\boldsymbol{x}$. In the appendix, we discuss HDMM+PGM, an extension of HDMM that enables it to scale more effectively in this setting.

Mechanisms. In the high-dimensional setting, there are far fewer data-independent mechanisms to choose from. We thus compare against Identity, Laplace, and Gaussian, which are the only methods from the previous section which are applicable and scalable to high-dimensional



(a) Time to run `SELECT` operators on 5-dimensional domains of size (c, c, c, c, c).

(b) Time to run `SELECT` operators on d-dimensional domains of size $(10, \ldots, 10)$.

(c) Time to run `RECONSTRUCT` for each type of strategy with varying domain sizes.

Figure 3. Scalability of different components of HDMM when run on multi-dimensional domains of varying size and shape.

settings. In addition to these simple baselines, we also compare against DataCube, which is applicable in this setting, but only for (unweighted) marginal query workloads.

Results and Findings. Table 6 and Table 7 report the RMSE of the baselines as well as each optimization operator. We compute the SVD bound when possible (i.e., the workload is either a single Kronecker product or a marginal query workload). We have four main findings which we enumerate below:

(1) HDMM is better than all competitors on all tasks, and the magnitude of the improvement is as large as 38 for Laplace noise and 29 for Gaussian noise.

(2) HDMM gets within a factor 1.06 of the SVD bound when it is possible to compute it for Gaussian noise. This is consistent with the theoretical result in Section 6 which justifies the defintion of $\mathsf{OPT}_\otimes$. For Laplace noise, however, the ratio is as high as 14.

(3) Gaussian noise offers lower error than Laplace noise for the two highest dimensional schemas, and comparable error for the two five-dimensional schemas. In contrast to the one-dimensional setting, this occurs because the savings from using the L_2 sensitivity norm outweighs the cost of $\approx \sqrt{\log(1/\delta)}$ to use Gaussian noise with (ϵ, δ)-differential privacy.

(4) The parameterization that offers the lowest error differs based on the workload and the type of noise added. For example, OPT_M is always the best for workloads consisting of Marginals, but it is also sometimes the best for other workloads too. $\mathsf{OPT}_\otimes$ is the best for the CPH and CPS workloads, but not as good for the Adult and Loans workloads. OPT_+ is best for the low-dimensional Prefix Marginals workloads.

Scalability. We now evaluate the scalability HDMM. The main factor that influences the scalability of HDMM is the domain. The optimization time primarily depends on the number of dimensions and the size of each dimension, while reconstruction time primarily depends on the total domain size. Thus, the bottleneck of HDMM depends on all of these factors in a nuanced way, and for some domains optimization will be the bottleneck, while for others reconstruction will be. We show how the key components scale with respect to these properties of the domain in Figure 3.

In Figure 3(a), we fix the number of dimensions of the domain at $d = 5$ and vary the size of each dimension from $n_i = 2$ to $n_i = 1024$. We measure and report the optimization time for $\mathsf{OPT}_\otimes, \mathsf{OPT}_+$, and OPT_M. We run $\mathsf{OPT}_\otimes$ for 100 inner iterations (in calls to OPT_0) and 5 outer iterations. We use a workload consisting of a union of 10 Kronecker products, where each subworkload is All Range queries. In Figure 3(b), we fix the domain size of each dimension at $n_i = 10$ and vary the number of dimensions from $d = 2$ to $d = 15$. We again use the same workload as before. In Figure 3(c), we use the strategies produced from Figure 3(a), and measure the time required to perform the `RECONSTRUCT` step of HDMM.

From the figure we can see that the optimization time of $\mathsf{OPT}_\otimes$ and OPT_+ primarily depends on the size of each dimension, rather than the number of dimensions. In contrast, the optimization time of OPT_M primarily depends on the number of dimensions and not the size of each dimension. This confirms the theoretical complexity results. All three optimization operators are capable of running in settings where the total domain size is far too large to allow $\boldsymbol{x}$ to be represented in vector form. The figure also shows that we can solve the `RECONSTRUCT` step up to domains as large as 10^9. Beyond this point, it is infeasible to even represent $\boldsymbol{x}$ in vector form on the machine used for experiments. Further scalability is possible by using the extension described in Section 10.

12. Related Work

Much research has been done to develop differentially private algorithms for accurately answering linear queries [2, 4, 10, 12, 24, 27–29, 31, 40–42, 42, 48–57]. These algorithms are either data-dependent (such as DAWA [27]) or data-independent (such as HB [41]). Hay et al. [23] found that in the high signal setting (number of records is large relative to ϵ and n), data-independent algorithms dominate, while in the low signal setting, data-dependent algorithms dominate. Most of the data-independent mechanisms belong to the **select-measure-reconstruct** paradigm, and much research has been done on the strategy selection problem for particular (usually fixed) workloads such as range queries or marginals. Some research has been done on the strategy selection problem that is automatically tuned to a user-specified workload. However, none of the existing approaches offer the scalability, generality, and utility of HDMM.

Mechanisms for Range Query Workloads. One notable class of workloads that has recieved considerable attention in the literature is range queries. For these workloads, Xiao et al. [48] propose a strategy based on wavelet transforms, Hay et al. [24] propose a hierarchical strategy, Cormode et al. [10] propose similar hierarchical strategies for multi-dimensional domains, and Qardaji et al. [41] generalize and improve the hierarchical approach. All of these strategies are designed for workloads of range queries, and they are not workload-adaptive. The HB approach proposed by Qardaji et al. [41] chooses a branching factor for a hierarchical strategy by optimizing an analytically computable approximation of TSE. Li et al. [27] propose an approach called GreedyH that is workload adaptive, but based on a template strategy designed for range query-like workloads. GreedyH can be seen as an instance of HDMM, since it optimizes over strategies parameterized by a small set of weights, but the parameterization is only reasonable for 1D range query workloads, and is not expressive enough to capture the strategies produced by our p-Identity parameterization.

Mechanisms for Marginal Query Workloads. Another notable class of workloads that has recieved special attention in the literature is marginal query workloads. Barak et al. [4] propose a strategy based on Fourier basis vectors for answering low-dimensional marginals over a binary domain. This method offers some workload adaptivity, in that the set of Fourier basis vectors in the strategy depends on the marginals in the workload.

Ding et al. [12] propose a strategy of marginals that adapts to the workload through a greedy heuristic that approximately solves a combinatoric optimization problem. This space of strategies considered by this approach is a subset of those representable by our marginals parameterization, where $\boldsymbol{\theta} \in \{0,1\}^{2^d}$. They give a method for efficiently doing least squares estimation for consistency, but unlike HDMM, their objective function doesn't account for this in the strategy selection phase.

In addition to the data-independent mechanisms listed above, there is a large body of work around mechanisms for answering marginal query workloads [13, 17, 21, 32, 42, 44, 47].

Workload-Adaptive Mechanisms. Some existing methods are truly workload-adaptive, such as the low rank mechanism [54] and COA [53]. These methods both rely on the matrix reperesentation of the workload and hence suffer from the same scalability limitations of the matrix mechanism. There are a few notable workload-adaptive mechanisms that do not rely on a matrix representation of the workload, however they do require some structural assumptions about the workload, just like HDMM assumes the workload contains conjunctive queries. Some notable examples include MWEM [21], DualQuery [17], and FEM [47]. These mechanisms are all data-dependent and can run for marginal query workloads.

13. Discussion and conclusions

In this paper, we introduce HDMM, a general and scalable method for privately answering collections of counting queries over high-dimensional data. HDMM is capable of running on multi-dimensional datasets with very large domains. This is primarily enabled by our implicit workload representation in terms of Kronecker products, and our optimization routines for strategy selection that exploit this implicit representation. Because HDMM provides state-of-the-art error rates in both low- and high-dimensions, and fully automated strategy selection, we believe it will be broadly useful to algorithm designers.

In this paper, we extend HDMM to handle Gaussian noise in addition to Laplace noise, showing that in several cases, strategy optimization is actually simpler and more effective. We also study the SVD bound with implicitly represented workloads, and used it to reason about the effectiveness of our optimization operators theoretically and empirically.

Acknowledgements: We would like to thank the two anonymous reviewers for providing thorough reviews and detailed comments to improve the clarity of this work, as well as providing ideas for improving the technical content. This work was supported by the National Science Foundation under grants 1253327, 1408982, 1409125, 1443014, 1421325, and 1409143; and by DARPA and SPAWAR under contract N66001-15-C-4067. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] J. M. Abowd. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2867–2867, 2018, pages 2867–2867. https://doi.org/10.1145/3219819.3220028.

[2] G. Ács, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *ICDM*, pages 1–10, 2012, pages 1–10. https://doi.org/10.1109/ICDM.2012.6459448.

[3] B. Balle and Y.-X. Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *International Conference on Machine Learning*, pages 394–403, 2018, pages 394–403. https://doi.org/10.1145/3219819.3220092.

[4] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 273–282. ACM, 2007, pages 273–282. https://doi.org/10.1145/1297454.1297481.

[5] A. Bhaskara, D. Dadush, R. Krishnaswamy, and K. Talwar. Unconditional differentially private mechanisms for linear queries. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1269–1284, New York, NY, USA, 2012. ACM, pages 1269–1284. URL: http://doi.acm.org/10.1145/2213977.2214089, https://doi.org/10.1145/2213977.2214089.

[6] M. Bun, K. Nissim, U. Stemmer, and S. Vadhan. Differentially private release and learning of threshold functions. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 634–649. IEEE, 2015, pages 634–649. https://doi.org/10.1109/FOCS.2015.634.

[7] M. Bun and T. Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*, pages 635–658. Springer, 2016, pages 635–658. https://doi.org/10.1007/978-3-319-39035-8_30.

[8] M. Bun, J. Ullman, and S. Vadhan. Fingerprinting codes and the price of approximate differential privacy. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 1–10, 2014, pages 1–10. https://doi.org/10.1145/2588996.2589001.

[9] 2010 Census Summary File 1, Census of Population and Housing. Available at https://www.census.gov/prod/cen2010/doc/sf1.pdf, 2012. https://doi.org/10.2307/3734763.

[10] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu. Differentially private spatial decompositions. In *Data engineering (ICDE), 2012 IEEE 28th international conference on*, pages 20–31. IEEE, 2012, pages 20–31. https://doi.org/10.1109/ICDE.2012.6195552.

[11] S. Denisov, B. McMahan, K. Rush, A. Smith, and A. Thakurta. Improved differential privacy for sgd via optimal private linear operators on adaptive streams. *arXiv preprint arXiv:2202.08312*, 2022. https://doi.org/10.48550/arXiv.2202.08312.

[12] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 217–228. ACM, 2011, pages 217–228. https://doi.org/10.1145/1989442.1989473.

[13] C. Dwork, A. Nikolov, and K. Talwar. Efficient algorithms for privately releasing marginals via convex relaxations. *Discrete & Computational Geometry*, 53(3):650–673, 2015. https://doi.org/10.1007/s00454-014-9603-z.

[14] C. Dwork, F. M. K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006, pages 265–284. https://doi.org/10.1145/1132927.1132943.

[15] C. Dwork and A. Roth. *The Algorithmic Foundations of Differential Privacy*. Found. and Trends in Theoretical Computer Science, 2014. https://doi.org/10.1561/200004039.

[16] A. Edmonds, A. Nikolov, and J. Ullman. The power of factorization mechanisms in local and central differential privacy. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 425–438, 2020, pages 425–438. https://doi.org/10.1145/3357713.3383008.

[17] M. Gaboardi, E. J. G. Arias, J. Hsu, A. Roth, and Z. S. Wu. Dual query: Practical private query release for high dimensional data. In *International Conference on Machine Learning*, pages 1170–1178. PMLR, 2014, pages 1170–1178. https://doi.org/10.5555/2968612.2968710.

[18] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012. https://doi.org/10.1137/110837277.

[19] W. W. Hager. Updating the inverse of a matrix. *SIAM review*, 31(2):221–239, 1989. https://doi.org/10.1137/1031019.

[20] S. Haney, A. Machanavajjhala, M. Kutzbach, M. Graham, J. Abowd, and L. Vilhuber. Utility cost of formal privacy for releasing national employer-employee statistics. In *ACM SIGMOD*, 2017. https://doi.org/10.1145/3035918.3035973.

[21] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. *arXiv preprint arXiv:1012.4763*, 2010. https://doi.org/10.1145/1821442.1821572.

[22] M. Hardt and K. Talwar. On the geometry of differential privacy. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 705–714, New York, NY, USA, 2010. ACM, pages 705–714. URL: http://doi.acm.org/10.1145/1806689.1806786, https://doi.org/10.1145/1806689.1806786.

[23] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, pages 139–154. ACM, 2016, pages 139–154. https://doi.org/10.1145/2889442.2889463.

[24] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1-2):1021–1032, 2010. https://doi.org/10.14778/1920405.1920442.

[25] HCUPnet: Healthcare Cost and Utilization Project. Available at `https://hcupnet.ahrq.gov/`.

[26] P. Lancaster and H. K. Farahat. Norms on direct sums and tensor products. *mathematics of computation*, 26(118):401–414, 1972. https://doi.org/10.2307/2004866.

[27] C. Li, M. Hay, G. Miklau, and Y. Wang. A data-and workload-aware algorithm for range queries under differential privacy. *PVLDB*, 7(5):341–352, 2014. https://doi.org/10.14778/2536200.2536236.

[28] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 123–134. ACM, 2010, pages 123–134. https://doi.org/10.1145/1807167.1807182.

[29] C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. *PVLDB*, 5(6):514–525, 2012. https://doi.org/10.14778/2211832.2211876.

[30] C. Li and G. Miklau. Optimal error of query sets under the differentially-private matrix mechanism. In *ICDT*, 2013. https://doi.org/10.1145/2484239.2484284.

[31] C. Li, G. Miklau, M. Hay, A. McGregor, and V. Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB Journal*, 24(6):757–781, 2015. https://doi.org/10.1007/s00778-015-0402-y.

[32] T. Liu, G. Vietri, and S. Z. Wu. Iterative methods for private synthetic data: Unifying framework and new methods. *Advances in Neural Information Processing Systems*, 34:690–702, 2021. https://doi.org/10.1016/j.aip.2021.10.038.

[33] A. Machanavajjhala, D. Kifer, J. M. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *ICDE*, pages 277–286, 2008, pages 277–286. https://doi.org/10.1109/ICDE.2008.4525277.

[34] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *Proceedings of the VLDB Endowment*, 11(10):1206–1219, 2018. https://doi.org/10.14778/3219819.3220028.

[35] R. Mckenna, D. Sheldon, and G. Miklau. Graphical-model based estimation and inference for differential privacy. In *International Conference on Machine Learning*, pages 4435–4444, 2019, pages 4435–4444. https://doi.org/10.1145/3295222.3330993.

[36] I. Mironov. R'enyi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE, 2017, pages 263–275. https://doi.org/10.1109/CSF.2017.16.

[37] A. Nikolov. An improved private mechanism for small databases. In *International Colloquium on Automata, Languages, and Programming*, pages 1010–1021. Springer, 2015, pages 1010–1021. https://doi.org/10.1007/978-3-319-19669-4_69.

[38] A. Nikolov, K. Talwar, and L. Zhang. The geometry of differential privacy: The sparse and approximate cases. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 351–360, New York, NY, USA, 2013. ACM, pages 351–360. URL: http://doi.acm.org/10.1145/2488608.2488652, https://doi.org/10.1145/2488608.2488652.

[39] OnTheMap Web Tool. Available at `http://onthemap.ces.census.gov/`.

[40] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *Intl. Conference on Data Engineering (ICDE)*, pages 757–768. IEEE, 2013, pages 757–768.

[41] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 6(14):1954–1965, 2013. https://doi.org/10.14778/2536200.2536236.

[42] W. Qardaji, W. Yang, and N. Li. Priview: practical differentially private release of marginal contingency tables. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1435–1446. ACM, 2014, pages 1435–1446. https://doi.org/10.1145/2588555.2594010.

[43] T. Steinke and J. Ullman. Between pure and approximate differential privacy. *arXiv preprint arXiv:1501.06095*, 2015. https://doi.org/10.1145/2670018.2670043.

[44] J. Thaler, J. Ullman, and S. Vadhan. Faster algorithms for privately releasing marginals. In *International Colloquium on Automata, Languages, and Programming*, pages 810–821. Springer, 2012, pages 810–821. https://doi.org/10.1007/978-3-642-31594-7_68.

[45] J. Vaidya, B. Shafiq, X. Jiang, and L. Ohno-Machado. Identifying inference attacks against healthcare data repositories. *AMIA Jt Summits Transl Sci Proc*, 2013, 2013. https://doi.org/10.1136/amiat-2013-011811.

[46] C. F. Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1):85–100, 2000. https://doi.org/10.1016/S0377-0427(99)00071-8.

[47] G. Vietri, G. Tian, M. Bun, T. Steinke, and S. Wu. New oracle-efficient algorithms for private synthetic data release. In *International Conference on Machine Learning*, pages 9765–9774. PMLR, 2020, pages 9765–9774. https://doi.org/10.1145/3707309.3707443.

[48] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1200–1214, 2011. https://doi.org/10.1109/TKDE.2011.122.

[49] Y. Xiao, L. Xiong, L. Fan, S. Goryczka, and H. Li. DPCube: Differentially private histogram release through multidimensional partitioning. *Transactions of Data Privacy*, 7(3), 2014. https://doi.org/10.1007/s12071-014-0133-7.

[50] J. Xu, Z. Zhang, X. Xiao, Y. Yang, and G. Yu. Differentially private histogram publication. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 32–43, 2012, pages 32–43. https://doi.org/10.1109/ICDE.2012.48.

[51] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *The VLDB Journal*, pages 1–26, 2013. URL: http://dx.doi.org/10.1007/s00778-013-0309-y, https://doi.org/10.1007/s00778-013-0309-y.

[52] G. Yaroslavtsev, G. Cormode, C. M. Procopiuc, and D. Srivastava. Accurate and efficient private release of datacubes and contingency tables. In *ICDE*, 2013. https://doi.org/10.1109/ICDE.2013.48.

[53] G. Yuan, Y. Yang, Z. Zhang, and Z. Hao. Convex optimization for linear query processing under approximate differential privacy. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2005–2014. ACM, 2016, pages 2005–2014. https://doi.org/10.1145/2939672.2939764.

[54] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Low-rank mechanism: optimizing batch queries under differential privacy. *PVLDB*, 5(11):1352–1363, 2012. https://doi.org/10.14778/2211832.2211884.

[55] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):25, 2017. https://doi.org/10.1145/2629398.

[56] J. Zhang, X. Xiao, and X. Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *SIGMOD*, 2016.

[57] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. Towards accurate histogram publication under differential privacy. In *SDM*, 2014. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611973440.68, arXiv:http://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.68, https://doi.org/10.1137/1.9781611973440.68.

Appendix A. Implicit Vectorization of Disjunctive Queries

HDMM can also optimize workloads containing disjunctive queries with no modification to the underlying optimization operators being necessary. The theorems below show how different logical operators on predicates impact the vector representation of the queries.

Theorem 15. *The vector representation of the negation $\neg\phi$ is $vec(\neg\phi) = \boldsymbol{T} - vec(\phi)$.*

Theorem 16. *The vector representation of the disjunction $\phi = \phi_A \vee \phi_B$ is,*

$$vec(\phi) = vec(\neg(\neg\phi_A \wedge \neg\phi_B)) = \boldsymbol{T} \otimes \boldsymbol{T} - (\boldsymbol{T} - vec(\phi_A)) \otimes (\boldsymbol{T} - vec(\phi_B)).$$

The theorem above uses DeMorgan's law to show that the disjunctive query can be converted into the negation of a conjunctive query. We can also similarly define a Cartesian product of disjunctive queries, as follows:

$$\mathbb{W} = \mathbf{1} \otimes \mathbf{1} - \boldsymbol{W}_1 \otimes \boldsymbol{W}_2,$$

where $\boldsymbol{W}_1 = \mathbf{1} - vec(\Phi_A)$ and $\boldsymbol{W}_2 = \mathbf{1} - vec(\Phi_B)$ and $\mathbf{1}$ is a matrix of ones having the same shape as $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ respectively. Thus, we can represent a Cartesian product of disjunctive queries as a *difference* of two Kronecker products.

Taking the gram matrix of $\mathbb{W}$ we observe:

$$\mathbb{W}^\top\mathbb{W} = (\mathbf{1}\otimes\mathbf{1})^\top(\mathbf{1}\otimes\mathbf{1}) - (\mathbf{1}\otimes\mathbf{1})^\top(\boldsymbol{W}_1\otimes\boldsymbol{W}_2) - (\boldsymbol{W}_1\otimes\boldsymbol{W}_2)^\top(\mathbf{1}\otimes\mathbf{1}) + (\boldsymbol{W}_1\otimes\boldsymbol{W}_2)^\top(\boldsymbol{W}_1\otimes\boldsymbol{W}_2).$$

Each term of the above expression simplifies to a single Kronecker product, so $\mathbb{W}^\top\mathbb{W}$ is actually a sum of four Kronecker products. Note that the standard conjunctive query workloads, containing a union of Kronecker products, also have a Gram matrix that is a sum of Kronecker products. Furthermore, the optimization operators only depend on $\mathbb{W}$ through $\mathbb{W}^\top\mathbb{W}$, and they expect the Gram matrix to be a sum of Kronecker products. Thus, they can run without modification on workloads having the above disjunctive form. Additionally, the workloads may contain arbitrary conbinations of conjunctive and disjunctive queries.

Appendix B. Missing Proofs

Theorem 3 (Norm of a Kronecker product)**.** *The following matrix norms decompose over the factors of the Kronecker product $\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$.*

$$\|\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d\|_{\mathcal{L}} = \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{\mathcal{L}}$$

$$\|\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d\|_{\mathcal{G}} = \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{\mathcal{G}}$$

$$\|\boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d\|_{F} = \prod_{i=1}^{d} \|\boldsymbol{A}_i\|_{F}$$

Proof. We prove these statements directly with algebraic manipulation:

$$\begin{aligned}
\|\boldsymbol{A} \otimes \boldsymbol{B}\|_{\mathcal{L}} &= \max_t \sum_q |\boldsymbol{A}(q_A, t_A)\boldsymbol{B}(q_B, t_B)| \\
&= \max_t \sum_q |\boldsymbol{A}(q_A, t_A)||\boldsymbol{B}(q_B, t_B)| \\
&= \max_{t_A} \sum_{q_A} |\boldsymbol{A}(q_A, t_A)| \max_{t_B} \sum_{q_B} |\boldsymbol{B}(q_B, t_B)| \\
&= \|\boldsymbol{A}\|_{\mathcal{L}} \|\boldsymbol{B}\|_{\mathcal{L}}
\end{aligned}$$

$$\begin{aligned}
\|\boldsymbol{A} \otimes \boldsymbol{B}\|_{\mathcal{G}}^2 &= \max_t \sum_q (\boldsymbol{A}(q_A, t_A)\boldsymbol{B}(q_B, t_B))^2 \\
&= \max_t \sum_q \boldsymbol{A}(q_A, t_A)^2 \boldsymbol{B}(q_B, t_B)^2 \\
&= \max_{t_A} \sum_{q_A} \boldsymbol{A}(q_A, t_A)^2 \max_{t_B} \sum_{q_B} \boldsymbol{B}(q_B, t_B)^2 \\
&= \|\boldsymbol{A}\|_{\mathcal{G}}^2 \|\boldsymbol{B}\|_{\mathcal{G}}^2
\end{aligned}$$

$$\begin{aligned}
\|\boldsymbol{A} \otimes \boldsymbol{B}\|_F^2 &= \sum_{q,t} (\boldsymbol{A}(q_A, t_A)\boldsymbol{B}(q_B, t_B))^2 \\
&= \sum_{q,t} \boldsymbol{A}(q_A, t_A)^2 \boldsymbol{B}(q_B, t_B)^2 \\
&= \sum_{q_A, t_A} \boldsymbol{A}(q_A, t_A)^2 \sum_{q_B, t_B} \boldsymbol{B}(q_B, t_B)^2 \\
&= \|\boldsymbol{A}\|_F^2 \|\boldsymbol{B}\|_F^2
\end{aligned}$$

□

Theorem 17 (Complexity of OPT_0)**.** *Given any p-Identity strategy $\boldsymbol{A}(\boldsymbol{\Theta})$, both the objective function $C(\boldsymbol{A}(\boldsymbol{\Theta}))$ and the gradient $\frac{\partial C}{\partial \boldsymbol{A}}$ can be evaluated in $O(pn^2)$ time.*

Proof. Assume $\boldsymbol{W}^\top \boldsymbol{W}$ has been precomputed and now express $\boldsymbol{A}^\top \boldsymbol{A}$ in terms of $\boldsymbol{\Theta}$ and $\boldsymbol{D}$:

$$\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{D}^\top \boldsymbol{D} + \boldsymbol{D}^\top \boldsymbol{\Theta}^\top \boldsymbol{\Theta} \boldsymbol{D} = \boldsymbol{D}[\boldsymbol{I}_n + \boldsymbol{\Theta}^\top \boldsymbol{\Theta}]\boldsymbol{D}.$$

Applying the identity $(\boldsymbol{X}\boldsymbol{Y})^{-1} = \boldsymbol{Y}^{-1}\boldsymbol{X}^{-1}$ together with the Woodbury identity [19] yields an expression for the inverse:

$$\begin{aligned}
(\boldsymbol{A}^\top \boldsymbol{A})^{-1} &= \boldsymbol{D}^{-1}[\boldsymbol{I}_n + \boldsymbol{\Theta}^\top \boldsymbol{\Theta}]^{-1} \boldsymbol{D}^{-1} \\
&= \boldsymbol{D}^{-1}[\boldsymbol{I}_n - \boldsymbol{\Theta}^\top (\boldsymbol{I}_p + \boldsymbol{\Theta}\boldsymbol{\Theta}^\top)^{-1} \boldsymbol{\Theta}] \boldsymbol{D}^{-1}.
\end{aligned}$$

We can compute $(\boldsymbol{A}^\top \boldsymbol{A})^{-1}(\boldsymbol{W}^\top \boldsymbol{W})$ in $O(n^2 p)$ time by evaluating the following expression from right to left:

$$(\boldsymbol{A}^\top \boldsymbol{A})^{-1}(\boldsymbol{W}^\top \boldsymbol{W}) = \boldsymbol{D}^{-2}(\boldsymbol{W}^\top \boldsymbol{W}) - \boldsymbol{D}^{-1}\boldsymbol{\Theta}^\top (\boldsymbol{I}_p + \boldsymbol{\Theta}\boldsymbol{\Theta}^\top)^{-1} \boldsymbol{\Theta} \boldsymbol{D}^{-1}(\boldsymbol{W}^\top \boldsymbol{W}).$$

By carefully looking at the dimensionality of the intermediate matrices that arise from carrying out the matrix multiplications from right-to-left, we see that the most expensive

operation is the matrix-matrix product between an $n \times p$ matrix and a $p \times n$ matrix, which takes $O(n^2 p)$ time. The inverse $(\boldsymbol{I}_p + \boldsymbol{\Theta}\boldsymbol{\Theta}^\top)^{-1}$ takes $O(p^3)$ time and the operations involving D take $O(n^2)$ time since it is a diagonal matrix.

The result still holds even if $\boldsymbol{W}^\top \boldsymbol{W}$ is replaced with an arbitrary $n \times n$ matrix, so $\boldsymbol{X} = (\boldsymbol{A}^\top \boldsymbol{A})^{-1}(\boldsymbol{W}^\top \boldsymbol{W})(\boldsymbol{A}^\top \boldsymbol{A})^{-1}$ can be computed in $O(n^2 p)$ time as well. The gradient is $-2\boldsymbol{A}\boldsymbol{X}$ whose components can be calculated separately as $-2\boldsymbol{D}\boldsymbol{X}$ and $-2\boldsymbol{\Theta}\boldsymbol{X}$. $-2\boldsymbol{D}\boldsymbol{X}$ takes $O(n^2)$ time and $\boldsymbol{\Theta}\boldsymbol{X}$ takes $O(n^2 p)$ time, so the overall cost of computing the gradient is $O(n^2 p)$. □

Theorem 7 (Multiplication of Marginal Gram Matrices)**.** *For any $a, b \in [2^d]$,*

$$\mathbb{H}(a)\mathbb{H}(b) = \boldsymbol{c}(a|b)\mathbb{H}(a\&b),$$

where $a|b$ denotes "bitwise or", $a\&b$ denotes "bitwise and", and $\boldsymbol{c}$ is the characteristic vector. Moreover, for any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{2^d}$,

$$\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) = \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{v}),$$

where $\boldsymbol{X}(\boldsymbol{u})$ is a $2^d \times 2^d$ triangular matrix with entries $\boldsymbol{X}(\boldsymbol{u})(k, b) = \sum_{a:a\&b=k} \boldsymbol{u}(a)\boldsymbol{c}(a|b)$.

Proof. First observe how the matrices $\boldsymbol{I}$ and $\boldsymbol{1}$ interact under matrix multiplication:

$$\boldsymbol{I}\boldsymbol{I} = \boldsymbol{I} \qquad \boldsymbol{I}\boldsymbol{1} = \boldsymbol{1} \qquad \boldsymbol{1}\boldsymbol{I} = \boldsymbol{1} \qquad \boldsymbol{1}\boldsymbol{1} = n_i \boldsymbol{1}.$$

Now consider the product $\mathbb{H}(a)\mathbb{H}(b)$ which is simplified using Kronecker product identities, logical rules, and bitwise manipulation.

$$\begin{aligned}
&= \bigotimes_{i=1}^{d} [\boldsymbol{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)][\boldsymbol{1}(b_i = 0) + \boldsymbol{I}(b_i = 1)] \\
&= \prod_{i=1}^{d} [n_i(a_i = 0 \text{ and } b_i = 0) + 1(a_i = 1 \text{ or } b_i = 1)] \bigotimes_{i=1}^{d} [\boldsymbol{1}(a_i = 0 \text{ or } b_i = 0) + \boldsymbol{I}(a_i = 1 \text{ and } b_i = 1)] \\
&= \prod_{i=1}^{d} [n_i((a|b)_i = 0) + 1((a|b)_i = 1)] \bigotimes_{i=1}^{d} [\boldsymbol{1}((a\&b)_i = 0) + \boldsymbol{I}((a\&b)_i = 1)] \\
&= \boldsymbol{c}(a|b)\mathbb{H}(a\&b)
\end{aligned}$$

Now let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{2^d}$ and consider the following product:

$$\begin{aligned}
\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) &= \Big(\sum_a \boldsymbol{u}(a)\mathbb{H}(a) \Big) \Big(\sum_b \boldsymbol{v}(b)\mathbb{H}(b) \Big) \\
&= \sum_{a,b} \boldsymbol{u}(a)\boldsymbol{v}(b)\mathbb{H}(a)\mathbb{H}(b) \\
&= \sum_{a,b} \boldsymbol{u}(a)\boldsymbol{v}(b)\boldsymbol{c}(a|b)\mathbb{H}(a\&b).
\end{aligned}$$

Observe that $\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) = \mathbb{G}(\boldsymbol{w})$ where

$$\boldsymbol{w}(k) = \sum_{a\&b=k} \boldsymbol{u}(a)\boldsymbol{v}(b)\boldsymbol{c}(a|b).$$

The relationship between $\boldsymbol{w}$ and $\boldsymbol{v}$ is clearly linear, and by carefully inspecting the expression one can see that $\boldsymbol{w} = \boldsymbol{X}(\boldsymbol{u})\boldsymbol{v}$ where $\boldsymbol{X}(\boldsymbol{u})(k, b) = \sum_{a:a\&b=k} \boldsymbol{u}(a)\boldsymbol{c}(a|b)$. $\boldsymbol{X}(\boldsymbol{u})$ is an upper triangular matrix because $k = a\&b$, and $a\&b \leq b$ for all a. □

Theorem 8 (Inverse of Marginal Gram Matrices)**.** *Let $\boldsymbol{X}(\boldsymbol{u})$ be the matrix defined in Theorem 7. If $\boldsymbol{X}(\boldsymbol{u})$ is invertible, then $\mathbb{G}(\boldsymbol{u})$ is invertible with inverse:*

$$\mathbb{G}^{-1}(\boldsymbol{u}) = \mathbb{G}(\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z}),$$

where $\boldsymbol{z}(2^d - 1) = 1$ and $\boldsymbol{z}(a) = 0$ for all other a. Moreover, if $\boldsymbol{X}^g(\boldsymbol{u})$ is a generalized inverse of $\boldsymbol{X}(\boldsymbol{u})$, then a generalized inverse of $\mathbb{G}(\boldsymbol{u})$ is given by,

$$\mathbb{G}^g(\boldsymbol{u}) = \mathbb{G}(\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}).$$

Proof. First note that $\mathbb{G}(\boldsymbol{z}) = \mathbb{I}$ (the identity matrix). By Theorem 7,

$$\begin{aligned}\mathbb{G}(\boldsymbol{u})\mathbb{G}^{-1}(\boldsymbol{u}) &= \mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z}) \\ &= \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^{-1}(\boldsymbol{u})\boldsymbol{z}) \\ &= \mathbb{G}(\boldsymbol{I}\boldsymbol{z}) = \mathbb{G}(\boldsymbol{z}) = \mathbb{I}.\end{aligned}$$

This proves the first part of the theorem statement. For the second part, note that if $\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v})\mathbb{G}(\boldsymbol{u}) = \mathbb{G}(\boldsymbol{u})$, then $\mathbb{G}(\boldsymbol{v})$ is a generalized inverse of $\mathbb{G}(\boldsymbol{u})$. Using $\boldsymbol{v} = \boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}$, we have,

$$\begin{aligned}\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v})\mathbb{G}(\boldsymbol{u}) &= \mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{v}) \\ &= \mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}) \\ &= \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}) \\ &= \mathbb{G}(\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}) \\ &= \mathbb{G}(\boldsymbol{I}\boldsymbol{X}(\boldsymbol{u})\boldsymbol{X}^g(\boldsymbol{u})\boldsymbol{u}) \\ &= \mathbb{G}(\boldsymbol{I}\boldsymbol{u}) = \mathbb{G}(\boldsymbol{u}).\end{aligned}$$

Thus, $\mathbb{G}(\boldsymbol{v})$ is a generalized inverse as desired. This completes the proof. □

Theorem 9 (Eigenvectors and Eigenvalues of Marginal Gram Matrices)**.** *Let $a \in [2^d]$ and let*

$$\mathbb{V}(a) = \bigotimes_{i=1}^{d} (a_i = 0)\boldsymbol{T} + (a_i = 1)(\boldsymbol{1} - n_i\boldsymbol{I}).$$

For any $b \in [2^d]$, $\mathbb{V}(a)$ is an eigenmatrix of $\mathbb{H}(b)$ with corresponding eigenvalue $\boldsymbol{\lambda}(a) = \boldsymbol{c}(b)$ if $a\&b = a$ and $\boldsymbol{\lambda}(a) = 0$ otherwise. Moreover, for any $\boldsymbol{w} \in \mathbb{R}^{2^d}$, $\mathbb{V}(a)$ is an eigenmatrix of $\mathbb{G}(\boldsymbol{w})$ with corresponding eigenvalue $\boldsymbol{\kappa}(a) = \sum_{b:a\&b=a} \boldsymbol{w}(b)\boldsymbol{c}(b)$. That is,

$$\mathbb{H}(b)\mathbb{V}(a) = \boldsymbol{\lambda}(a)\mathbb{V}(a), \qquad \mathbb{G}(\boldsymbol{w})\mathbb{V}(a) = \boldsymbol{\kappa}(a)\mathbb{V}(a).$$

Proof. Recall that $\mathbb{H}(b) = \bigotimes_{i=1}^{d}[\boldsymbol{1}(b_i = 0) + \boldsymbol{I}(b_i = 1)]$ and $\boldsymbol{c}(k) = \prod_{i=1}^{d}[n_i(k_i = 0) + 1(k_i = 1)]$. The proof follows from direct calculation:

$$\begin{aligned}
\mathbb{H}(b)\mathbb{V}(a) &= \bigotimes_{i=1}^{d}[(b_i=0)\mathbf{1}+(b_i=1)\boldsymbol{I}]\bigotimes_{i=1}^{d}[(a_i=0)\boldsymbol{T}+(a_i=1)(\mathbf{1}-n_i\boldsymbol{I})] \\
&= \bigotimes_{i=1}^{d}[(b_i=0)\mathbf{1}+(b_i=1)\boldsymbol{I}][(a_i=0)\boldsymbol{T}+(a_i=1)(\mathbf{1}-n_i\boldsymbol{I})] \\
&= \bigotimes_{i=1}^{d}[(a_i=0 \text{ and } b_i=0)n_i\boldsymbol{T}+(a_i=0 \text{ and } b_i=1)\boldsymbol{T} \\
&+ (a_i=1 \text{ and } b_i=0)\mathbf{0}+(a_i=1 \text{ and } b_i=1)(\mathbf{1}-n_i\boldsymbol{I})] \\
&= \begin{cases} \prod_{i=1}^{d} n_i(b_i=0)+1(b_i=1)\bigotimes_{i=1}^{d}[(a_i=0)\boldsymbol{T}+(a_i=1)(\mathbf{1}-n_i\boldsymbol{I})] & a\&b=a \\ \mathbb{0} & otherwise \end{cases} \\
&= \begin{cases} \boldsymbol{c}(b)\mathbb{V}(a) & a\&b=a \\ 0\mathbb{V}(a) & otherwise \end{cases} \\
&= \boldsymbol{\lambda}(a)\mathbb{V}(a)
\end{aligned}$$

This completes the first part of the proof. For the second part, we have,

$$\begin{aligned}
\mathbb{G}(\boldsymbol{w})\mathbb{V}(a) &= \sum_b \boldsymbol{w}(b)\mathbb{H}(b)\mathbb{V}(a) \\
&= \sum_b \boldsymbol{w}(b)\boldsymbol{\lambda}(a)\mathbb{V}(a) \\
&= \sum_{b:a\&b=a} \boldsymbol{w}(b)C(b)\mathbb{V}(a) \\
&= \boldsymbol{\kappa}(a)\mathbb{V}(a).
\end{aligned}$$

□

Theorem 10 (Marginal approximation of conjunctive query workload)**.** *For any conjunctive query workload* $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$*, there is a marginal Gram matrix* $\mathbb{G}(\boldsymbol{w})$[11] *such that* $tr[\mathbb{G}(\boldsymbol{u})\mathbb{W}^\top\mathbb{W}] = tr[\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{w})]$ *for all* $\boldsymbol{u}$*.*

Proof. Let $\mathbb{V} = \mathbb{W}^\top\mathbb{W}$ be the Gram matrix of $\mathbb{W}$: $\mathbb{V} = \sum_{j=1}^{k} w_j^2 \bigotimes_{i=1}^{d} \boldsymbol{V}_i^{(j)}$ where $\boldsymbol{V}_i^{(j)} = (\boldsymbol{W}^\top\boldsymbol{W})_i^{(j)}$. Now consider the following quantity:

[11] $\boldsymbol{w}$ is related to, but not equal to $w_1, \ldots, w_k$; $\boldsymbol{w}$ has size $2^d \neq k$.

$$
\begin{aligned}
tr[\mathbb{G}(\boldsymbol{u})\mathbb{V}] &= tr\Big[\Big(\sum_{a=0}^{2^d-1} \boldsymbol{u}(a) \bigotimes_{i=1}^{d} [\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)]\Big)\Big(\sum_{j=1}^{k} w_j^2 \bigotimes_{i=1}^{d} \boldsymbol{V}_i^{(j)}\Big)\Big] \\
&= tr\Big[\sum_{a=0}^{2^d-1} \boldsymbol{u}(a) \sum_{j=1}^{k} w_j^2 \bigotimes_{i=1}^{d} [\mathbf{1}(a_i = 0) + \boldsymbol{I}(a_i = 1)]\boldsymbol{V}_i^{(j)}\Big] \\
&= \sum_{a=0}^{2^d-1} \boldsymbol{u}(a) \sum_{j=1}^{k} w_j^2 \prod_{i=1}^{d} tr[\mathbf{1}\boldsymbol{V}_i^{(j)}](a_i = 0) + tr[\boldsymbol{I}\boldsymbol{V}_i^{(j)}](a_i = 1) \\
&= \sum_{a=0}^{2^d-1} \boldsymbol{u}(a) \sum_{j=1}^{k} w_j^2 \prod_{i=1}^{d} sum[\boldsymbol{V}_i^{(j)}](a_i = 0) + tr[\boldsymbol{V}_i^{(j)}](a_i = 1)
\end{aligned}
$$

Observe that it only depends on $\boldsymbol{V}_i^{(j)}$ through its *sum* and *trace*. Thus, we could replace $\boldsymbol{V}_i^{(j)}$ with any matrix that has the same *sum* and *trace*. In particular, we could use $\hat{\boldsymbol{V}}_i^{(j)} = b\boldsymbol{I} + c\mathbf{1}$, where b and c are chosen to satisfy the following linear system:

$$
\begin{bmatrix} n_i & n_i \\ n_i & n_i^2 \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix} = \begin{bmatrix} tr[\boldsymbol{V}_i^{(j)}] \\ sum[\boldsymbol{V}_i^{(j)}] \end{bmatrix}.
$$

The matrix $\hat{\mathbb{V}}_j = w_j^2(\hat{\boldsymbol{V}}_1^{(j)} \otimes \cdots \otimes \hat{\boldsymbol{V}}_d^{(j)})$ is nothing more than the Gram matrix for a collection of weighted marginals, or $\mathbb{G}(\boldsymbol{w}_j)$. This is because each factor in the Kronecker product is a weighted sum of $\boldsymbol{I}$ and $\mathbf{1}$, and by using the distributive property it can be converted into the canonical representation.

Thus, the matrix $\sum_j \mathbb{G}(\boldsymbol{w}_j) = \mathbb{G}(\sum_j \boldsymbol{w}_j) = \mathbb{G}(\boldsymbol{w})$ satisfies $tr[\mathbb{G}(\boldsymbol{u})\mathbb{V}] = tr[\mathbb{G}(\boldsymbol{u})\mathbb{G}(\boldsymbol{w})]$ as desired. □

Theorem 11 (Marginal parameterization objective function)**.** *Let $\mathbb{W} = w_1\mathbb{W}_1 + \cdots + w_k\mathbb{W}_k$ be a conjunctive query workload and let $\mathbb{G}(\boldsymbol{w})$ be the marginal approximation of $\mathbb{W}^\top\mathbb{W}$ (as in Theorem 10). For any marginal query strategy $\mathbb{A} = \mathbb{M}(\boldsymbol{\theta})$, the matrix mechanism objective function can be expressed as,*

$$
\|\mathbb{A}\|_{\mathcal{K}}^2 \left\|\mathbb{W}\mathbb{A}^+\right\|_F^2 = \|\boldsymbol{\theta}\|_{\mathcal{K}}^2 [\mathbf{1}^\top \boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w}].
$$

where $\|\boldsymbol{\theta}\|_{\mathcal{K}}$ is the sensitivity norm defined in Proposition 7, and $\boldsymbol{X}$ is the matrix defined in Theorem 7.

Proof.

$$\begin{aligned}
\|\mathbb{M}(\boldsymbol{\theta})\|_{\mathcal{K}}^2 \left\|\mathbb{W}\mathbb{M}(\boldsymbol{\theta})^+\right\|_F^2 &= \|\boldsymbol{\theta}\|^2 \left\|\mathbb{W}\mathbb{M}(\boldsymbol{\theta})^+\right\|_F^2 && \text{by Proposition 7} \\
&= \|\boldsymbol{\theta}\|^2 \, tr[\mathbb{G}^+(\boldsymbol{\theta}^2)\mathbb{W}^\top\mathbb{W}] \\
&= \|\boldsymbol{\theta}\|^2 \, tr[\mathbb{G}^+(\boldsymbol{\theta}^2)\mathbb{G}(\boldsymbol{w})] && \text{by Theorem 10} \\
&= \|\boldsymbol{\theta}\|^2 \, tr[\mathbb{G}(\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{\theta}^2)\mathbb{G}(\boldsymbol{w})] && \text{by Theorem 8} \\
&= \|\boldsymbol{\theta}\|^2 \, tr[\mathbb{G}(\boldsymbol{X}(\boldsymbol{w})\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{\theta}^2)] && \text{by Theorem 7} \\
&= \|\boldsymbol{\theta}\|^2 \, tr[\mathbb{G}(\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{X}(\boldsymbol{\theta}^2)\boldsymbol{w})] && \text{by commutativity} \\
&= \|\boldsymbol{\theta}\|^2 \, tr[\mathbb{G}(\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w})] && \text{by constraint} \\
&= \|\boldsymbol{\theta}\|^2 \, [\mathbf{1}^\top\boldsymbol{X}^+(\boldsymbol{\theta}^2)\boldsymbol{w}]
\end{aligned}$$

□

Theorem 12 (SVD Bound for Marginal Query Workloads)**.** *The SVD bound for a marginal query workload* $\mathbb{W}$ *with Gram matrix* $\mathbb{G}(\boldsymbol{w})$ *is,*

$$SVDB(\mathbb{W}) = \frac{1}{n}\Big(\sum_a \boldsymbol{c}(\neg a)\sqrt{\sum_{b:a\&b=a} \boldsymbol{w}(b)\boldsymbol{c}(b)}\Big)^2.$$

Proof. From Theorem 9 we know all 2^d unique eigenvalues and corresponding eigenmatrices. The number of rows in each eigenmatrix corresponds to the number of eigenvectors with that eigenvalue. To compute the SVD bound, we need to take the square root of each unique eigenvalue (which is a singular value of $\mathbb{W}$) and multiply that by it's multiplicity, then sum across all unique eigenvalues. Note that the eigenmatrix $\mathbb{V}(a)$ has $\boldsymbol{c}(\neg a)$ rows. Hence, the SVD bound is:

$$\begin{aligned}
SVDB(\mathbb{W}) &= \frac{1}{n}\Big(\sum_a \boldsymbol{c}(\neg a)\sqrt{\boldsymbol{\kappa}(a)}\Big)^2 \\
&= \frac{1}{n}\Big(\sum_a \boldsymbol{c}(\neg a)\sqrt{\sum_{b:a\&b=a} \boldsymbol{w}(b)\boldsymbol{c}(b)}\Big)^2.
\end{aligned}$$

□

Theorem 13 (Closed form solution to Problem 2)**.** *Let* $\mathbb{W}$ *be a workload with Gram matrix* $\mathbb{G}(\boldsymbol{w})$ *and let* $\boldsymbol{\theta} = \sqrt{\boldsymbol{Y}^{-1}\sqrt{\boldsymbol{Y}\boldsymbol{w}}}$ *(element-wise square root), where* $\boldsymbol{Y}$ *is the* $2^d \times 2^d$ *matrix:*

$$\boldsymbol{Y}(a,b) = \begin{cases} \boldsymbol{c}(b) & a\&b = a \\ 0 & otherwise \end{cases}.$$

If $\boldsymbol{\theta}$ *contains real-valued entries then the strategy* $\mathbb{A} = \mathbb{M}(\boldsymbol{\theta})$ *attains the SVDB bound when* $\mathcal{K} = \mathcal{G}$*, and is thus an optimal strategy. That is,* $\|\mathbb{A}\|_{\mathcal{G}}^2 \left\|\mathbb{W}\mathbb{A}^+\right\|_F^2 = SVDB(\mathbb{W})$.

Proof. We will prove optimality by showing that $\mathbb{A} = \mathbb{M}(\boldsymbol{\theta})$ matches the SVD bound. Li et al. [30] showed that the SVD bound is satisfied with equality if $\mathbb{A}$ and $\mathbb{W}$ share the same singular vectors and the singular values of $\mathbb{A}$ are the square root of the singular values of $\mathbb{W}$, at least in the case of Gaussian noise. Recall from Theorem 9 we know that all marginal Gram matrices share the same eigenvectors. The unique eigenvalues of $\mathbb{G}(\boldsymbol{w})$ are $\boldsymbol{\kappa} = \boldsymbol{Y}\boldsymbol{w}$. The gram matrix of $\mathbb{A} = \mathbb{M}(\boldsymbol{\theta})$ is $\mathbb{A}^\top\mathbb{A} = \mathbb{G}(\boldsymbol{\theta}^2)$. The eigenvalues of this are

$\boldsymbol{Y}\boldsymbol{\theta}^2 = \boldsymbol{Y}(\boldsymbol{Y}^{-1}\sqrt{\boldsymbol{Y}\boldsymbol{w}}) = \sqrt{\boldsymbol{Y}\boldsymbol{w}}$. Thus, the eigenvalues are exactly the square root of the eigenvalues of $\mathbb{G}(\boldsymbol{w})$, as desired. This certifies that $\mathbb{A} = \mathbb{M}(\boldsymbol{\theta})$ matches the SVD bound and is optimal. □

Theorem 14 (Efficient matrix-vector multiplication)**.** *Let $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \boldsymbol{A}_d$ and let $\boldsymbol{x}$ be a data vector of compatible shape. Then Algorithm 2 computes the matrix-vector product $\mathbb{A}\boldsymbol{x}$. Furthermore, if $\boldsymbol{A}_i \in \mathbb{R}^{n_i \times n_i}$ and $n = \prod n_i$ is the size of $\boldsymbol{x}$ then Algorithm 2 runs in $O(n \sum n_i)$ time.*

Proof. Let $\boldsymbol{y} = \mathbb{A}\boldsymbol{x}$. Then,

$$\begin{aligned}\boldsymbol{y}(q) &= \sum_t \mathbb{A}(q,t)\boldsymbol{x}(t) \\ &= \sum_t \boldsymbol{A}_1(q_1,t_1)\ldots\boldsymbol{A}_d(q_d,t_d)\boldsymbol{x}(t) \\ &= \sum_{t_1} \boldsymbol{A}_1(q_1,t_1)\cdots\sum_{t_d}\boldsymbol{A}_d(q_d,t_d)\boldsymbol{x}(t_1,\ldots,t_d).\end{aligned}$$

Now define $\boldsymbol{f}_k$ to be the vector indexed by tuples $(t_1,\ldots,t_{k-1},q_k,\ldots,q_d)$ such that $\boldsymbol{f}_{d+1} = \boldsymbol{x}$ and

$$\boldsymbol{f}_k(t_{1:k-1},q_{k:d}) = \sum_{t_k}\boldsymbol{A}_k(q_k,t_k)\boldsymbol{f}_{k+1}(t_{1:k},q_{k+1:d}),$$

and observe that $\boldsymbol{y} = \boldsymbol{f}_1$. We can efficiently compute $\boldsymbol{f}_k$ from $\boldsymbol{f}_{k+1}$ by observing that it is essentially computing a matrix-matrix product between the $n_k \times n_k$ matrix $\boldsymbol{A}_k$ and the $n_k \times n/n_k$ matrix obtained by reorganizing the entries of $\boldsymbol{f}_{k+1}$ into a matrix where rows are indexed by t_k. This can be computed in $O(nn_k)$ time. Thus, the total time required to compute $\boldsymbol{y}$ is $O(n\sum n_i)$ as stated. □

Appendix C. HDMM+PGM

Thus far, HDMM has addressed the fundamental limitation of the matrix matrix mechanism — replacing explicit matrix representations with implicit ones, and deriving efficient algorithms to solve the strategy optimization problem in the implicit space. Our innovations allow HDMM to run in much higher-dimensional settings than the matrix mechanism, but HDMM still has trouble scaling to very high-dimensional settings, when the data vector no longer fits in memory. Representing the data in vector form requires storing $n = \prod n_i$ entries, which grows exponentially with the number of dimensions, and quickly becomes infeasible for truly high-dimensional data. For example, a 30-dimensional dataset with binary attributes ($n_i = 2$) would require storing a data vector with 2^{30} entries, which is equivalent to approximately 4 GB of space. Scaling beyond this point would be quite challenging for HDMM.

It is important to note that the bottleneck of HDMM is `MEASURE` and `RECONSTRUCT`, as these steps access and estimate the data vector. In the matrix mechanism the main bottleneck is `SELECT`, as strategy optimization is the most expensive step. HDMM can often still perform the `SELECT` step efficiently even when `MEASURE` and `RECONSTRUCT` are intractable. In some special-but-common cases, it may be possible for HDMM to bypass this bottleneck on `MEASURE` and `RECONSTRUCT`, even scaling to settings where the data vector no longer fits in memory, making it suitable for arbitrarily large domains.

The settings where HDMM can bypass this limitation depends crucially on the strategy, and consequently the workload as well. If the workload is Identity over the whole domain (or any other full rank workload), then very little can be done because the vector of workload query answers (the output of HDMM) is just as large as the data vector itself, and simply enumerating those answers would require too much space. Thus, the number of queries in the workload cannot be too large. A special-but-common case occurs when the workload contains conjunctive queries over *small subsets of attributes*. The workload may cover all attributes of the dataset, but it will generally consist of a number of subworkloads, each which only cover a handful of attributes at a time. With workloads of this form, strategies produced by HDMM (OPT_+ and OPT_M in particular[12]) will generally contain queries that are also defined over small subsets of attributes. When this is the case, `MEASURE` can be done by keeping the data in its natural tabular format, and only vectorizing the data with respect to the relevant attributes for each sub-workload or sub-strategy. Since these are assumed to be defined over small subsets of attributes, these smaller data vectors can easily be materialized explicitly and operated on accordingly. Thus, the main remaining challenge is to `RECONSTRUCT` the workload query answers while avoiding an explicit representation for $\hat{\boldsymbol{x}}$. This can be done using a recently developed technique for efficient inference in differential privacy called "`Private-PGM`" [35]. `Private-PGM` consumes as input a set of noisy measurements defined over low-dimensional marginals, and produces a compact implicit representation of $\hat{\boldsymbol{x}}$. It leverages *probabilistic graphical models* to compactly represent $\hat{\boldsymbol{x}}$ in terms of a product of low-dimensional factors, and is able to scale to arbitrarily large domains as long as the measurements allow it.

Using `Private-PGM` with HDMM does change the mechanism in some subtle but important ways. The default HDMM method for `RECONSTRUCT` is based on standard ordinary least squares, as it computes $\hat{\boldsymbol{x}} = \mathbb{A}^+\boldsymbol{y}$, which is the solution to the minimization problem $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\mathbb{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$. In contrast, `Private-PGM` is based on the related non-negative least squares problem: $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}>0} \|\mathbb{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$. We know that the true data vector is non-negative, so for this reason it seems like the `Private-PGM` approach is more natural. However, non-negativity comes at the cost of *bias*. An appealing property of the ordinary least squares solution is that it produces an unbiased estimate of $\boldsymbol{x}$ under mild conditions. Non-negative least squares does not share this same guarantee. However, the introduction of bias often comes with reduced variance, and overall error is usually better when enforcing non-negativity [31, 35]. Thus `Private-PGM` can be used not only to improve scalability of HDMM, but also utility. In practical settings where some bias can be tolerated for reduced variance, we generally recommend incorporating `Private-PGM` post-processing into HDMM to improve utility, even when it is not necessary for scalability reasons.

Appendix D. Optimality of the Kronecker Product Parameterization

In Section 6.1 we motivated the use of the Kronecker parameterization by showing that when $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$ (the workload is a single Kronecker product), we can find a Kronecker product strategy $\mathbb{A} = \boldsymbol{A}_1 \otimes \cdots \otimes \ldots \boldsymbol{A}_d$ efficiently by invoking $\boldsymbol{A}_i = \mathsf{OPT}_0(\boldsymbol{W}_i)$. Moreover, when $\boldsymbol{A}_i$ attains the SVD Bound for $\boldsymbol{W}_i$ for all i, then $\mathbb{A}$ achieves the SVD bound for workload $\mathbb{W}$, which provides a certificate of optimality for $\mathbb{A}$. It would be nice if we

[12]Strategies produced by $\mathsf{OPT}_\otimes$ will generally be defined over the whole domain, rather than over a small subset of attributes.

could make the stronger claim that there is *always* an optimal strategy that is a Kronecker product for this type of workload. It turns out that this stronger claim is true, at least in the L_2 (Gaussian) version of the mechanism.

Let $\boldsymbol{X}$ denote the optimizer of Definition 12 for an explicilty represented workload $\boldsymbol{W}$. Then by [11] (Theorem 3.2), there is a vector of dual variables $\boldsymbol{v} \geq \boldsymbol{0}$ such that

$$\boldsymbol{W}^\top \boldsymbol{W} = \boldsymbol{X} diag(\boldsymbol{v}) \boldsymbol{X}.$$

The existence of such a $\boldsymbol{v}$ provides a certificate of optimality for $\boldsymbol{X}$. With that in mind, consider $\mathbb{W} = \boldsymbol{W}_1 \otimes \cdots \otimes \boldsymbol{W}_d$ and let $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_d$ be the optimizers of $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_d$ with corresponding dual variables $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d$. Using this fact, it is now easy to show that $\mathbb{X} = \boldsymbol{X}_1 \otimes \cdots \otimes \boldsymbol{X}_d$ is optimal for $\mathbb{W}$ with dual variables $\boldsymbol{v} = \boldsymbol{v}_1 \otimes \cdots \otimes \boldsymbol{v}_d$, since,

$$\mathbb{W}^\top \mathbb{W} = \bigotimes_{i=1}^{d} \boldsymbol{W}_i^\top \boldsymbol{W}_i,$$

and

$$\mathbb{X} diag(\boldsymbol{v}) \mathbb{X} = \bigotimes_{i=1}^{d} \boldsymbol{X}_i diag(\boldsymbol{v}_i) \boldsymbol{X}_i.$$

These two quantities are equal because $\boldsymbol{W}_i^\top \boldsymbol{W}_i = \boldsymbol{X}_i diag(\boldsymbol{v}_i) \boldsymbol{X}_i$. Hence, $\mathbb{X}$ is optimal for $\mathbb{W}$. Moreover, $\mathbb{A} = \bigotimes_{i=1}^{d} \boldsymbol{A}_i$ can be recovered by Cholesky decomposition, i.e., $\boldsymbol{A}_i^\top \boldsymbol{A}_i = \boldsymbol{X}_i$.