In-Database Regression in Input Sparsity Time

Rajesh Jayaram [1] **Alireza Samadian** [2] **David P. Woodruff** [1] **Peng Ye** [3]

Abstract

Sketching is a powerful dimensionality reduction technique for accelerating algorithms for data analysis. A crucial step in sketching methods is to compute a *subspace embedding* (SE) for a large matrix $\mathbf{A} \in \mathbb{R}^{N \times d}$. SE's are the primary tool for obtaining extremely efficient solutions for many linear-algebraic tasks, such as least squares regression and low rank approximation. Computing an SE often requires an explicit representation of $\mathbf{A}$ and running time proportional to the size of $\mathbf{A}$. However, if $\mathbf{A} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m$ is the result of a *database join query* on several smaller tables $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$, then this running time can be prohibitive, as $\mathbf{A}$ itself can have as many as $O(n_1 n_2 \cdots n_m)$ rows. In this work, we design subspace embeddings for database joins which can be computed significantly faster than computing the join. For the case of a two table join $\mathbf{A} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ we give *input-sparsity* algorithms for computing subspace embeddings, with running time bounded by the number of non-zero entries in $\mathbf{T}_1, \mathbf{T}_2$. This results in input-sparsity time algorithms for high accuracy regression, significantly improving upon the running time of prior FAQ-based methods for regression. We extend our results to arbitrary joins for the ridge regression problem, also considerably improving the running time of prior methods. Empirically, we apply our method to real datasets and show that it is significantly faster than existing algorithms.

[1]Computer Science Department, Carnegie Mellon University, Pittsburgh PA, United States. [2]Department of Computer Science, University of Pittsburgh, Pittsburgh PA, United States. [3]Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China.. Correspondence to: David P. Woodruff <dwoodruf@cs.cmu.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021.

1. Introduction

Sketching is an important tool for dimensionality reduction, whereby one quickly reduces the size of a large-scale optimization problem while approximately preserving the solution space. One can then solve the lower-dimensional problem much more efficiently, and the sketching guarantee ensures that the resulting solution is approximately optimal for the original optimization problem. In this paper we focus on the notion of a *subspace embedding* (SE), and its applications to problems in databases. Formally, given a large matrix $\mathbf{A} \in \mathbb{R}^{N \times d}$, an ϵ-subspace embedding for $\mathbf{A}$ is a matrix $\mathbf{SA}$, where $\mathbf{S} \in \mathbb{R}^{k \times N}$, with the property that $\|\mathbf{SA}x\|_2 = (1 \pm \epsilon)\|\mathbf{A}x\|_2$ simultaneously for all $x \in \mathbb{R}^d$.

A prototypical example of how a subspace embedding can be applied to solve an optimization problem is linear regression, where one wants to solve $\min_x \|\mathbf{A}x - b\|_2$ for a tall matrix $\mathbf{A} \in \mathbb{R}^{N \times d}$ where $N \gg d$ contains many data points (rows). Instead of directly solving for x, which requires computing the covariance matrix of $\mathbf{A}$ and which would require $O(Nd^2)$ time for general $\mathbf{A}$[1], one can first compute a *sketch* $\mathbf{SA}, \mathbf{S}b$ of the problem, where $\mathbf{S} \in \mathbb{R}^{k \times N}$ is a random matrix which can be quickly applied to $\mathbf{A}$ and b. If $[\mathbf{SA}, \mathbf{S}b]$ is an ϵ-subspace embedding for $[\mathbf{A}, b]$, it follows that the regression problem using the solution $\hat{x}$ to $\min_x \|\mathbf{SA}x - \mathbf{S}b\|_2$ will be within a $(1+\epsilon)$ factor of the optimal solution cost. However, if $k \ll N$, then solving for $\hat{x}$ can now be accomplished much faster – in $O(kd^2)$ time. SEs and similar tools for dimensionality reduction can also be used to speed up the running time of algorithms for ℓ_p regression, low rank approximation, and many other problems. We refer the reader to the survey (Woodruff et al., 2014) for a survey of applications of sketching to randomized numerical linear algebra.

One potential downside of most standard subspace embeddings is that the time required to compute $\mathbf{SA}$ often scales linearly with the *input sparsity* of $\mathbf{A}$, meaning the number of non-zero entries of $\mathbf{A}$, which we denote by $\texttt{nnz}(\mathbf{A})$. This dependence on $\texttt{nnz}(\mathbf{A})$ is in general necessary just to read all of the entries of $\mathbf{A}$. However, in many applications the dataset $\mathbf{A}$ is highly structured, and is itself the result of a query performed on a much smaller dataset.

[1]This can be sped up to $O(Nd^{\omega-1})$ time in theory, where $\omega \approx 2.373$ is the exponent of matrix multiplication.

A canonical and important example of this is a database join. This example is particularly important since datasets are often the result of a database join (Hellerstein et al., 2012). In fact, this use-case has motivated companies and teams such as RelationalAI (Rel) and Google's Bigquery ML (Big) to design databases that are capable of handling machine learning queries. Here, we have m tables $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_m$, where $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$, and we can consider their join $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m \in \mathbb{R}^{N \times d}$ over a set of columns. In general, the number N of rows of $\mathbf{J}$ can be as large as $n_1 n_2 \cdots n_m$, which far exceeds the actual input description of $\sum_i \mathtt{nnz}(\mathbf{T}_i)$ to the problem.

We note that it is possible to do various operations on a join in sublinear time using database algorithms that are developed for Functional Aggregation Queries (FAQ) (Abo Khamis et al., 2016), and indeed it is possible using so-called FAQ-based techniques (Abo Khamis et al., 2018) to compute the covariance matrix $\mathbf{J}^T\mathbf{J}$ in time $O(d^4 mn \log(n))$ for an acyclic join, where $n = \max(n_1, \ldots, n_m)$, after which one can solve least squares regression in poly(d) time. While this can be significantly faster than the $\Omega(N)$ time required to compute the actual join, it is still significantly larger than the input description size $\sum_i \mathtt{nnz}(\mathbf{T}_i)$, which even if the tables are dense, is at most dmn. When the tables are sparse, e.g., $O(n)$ non-zero entries in each table, one could even hope for a running time close to $O(mn)$. One could hope to achieve such running times with the help of subspace embeddings, by first reducing the data to a low-dimensional representation, and then solving regression exactly on the small representation. However, due to the lack of a clean algebraic structure for database joins, it is not clear how to apply a subspace embedding without first computing the join. Thus, a natural question is:

Is it possible to apply a subspace embedding to a join, without having to explicitly form the join?

We note that the lack of input-sparsity time algorithms for regression on joins is further exacerbated in the presence of categorical features. Indeed, it is a common practice to convert categorical data to their so-called *one-hot encoding* before optimizing any statistical model. Such an encoding creates one column for each possible value of the categorical feature and only a single column is non-zero for each row. Thus, in the presence of categorical features, the tables in the join are extremely high dimensional and extremely sparse. Since the data is high-dimensional, one often regularizes it to avoid overfitting, and so in addition to ordinary regression, one could also ask to solve regularized regression on such datasets, such as ridge regression. One could then ask if it is possible to design input-sparsity time algorithms on joins for regression or ridge regression.

1.1. Our Contributions

We start by describing our results for least squares regression in the important case when the join is on two tables. We note that the two-table case is a very well-studied case, see, e.g., (Alon et al., 1999; 2002; Gucht et al., 2015). Moreover, one can always reduce to the case of a two-table join by precomputing part of a general join. The following two theorems state our results for computing a subspace embedding and for solving regression on the join $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ of two tables. Our results demonstrate a substantial improvement over all prior algorithms for this problem. In particular, they answer the two questions above, showing that it is possible to compute a subspace embedding in input-sparsity time, and that regression can also be solved in input-sparsity time.

To the best of our knowledge, the fastest algorithm for linear regression on two tables has a worst-case time complexity of $\tilde{O}(nd + nD^2)$, where $n = \max(n_1, n_2)$, d is the number of columns, and D is the dimensionality of the data after encoding the categorical data. Note that in the case of numerical data (dense case) $D = d$ since there is no one-hot encoding and the time complexity is $O(nd^2)$, and it can be further improved to $O(nd^{\omega-1})$ where $\omega < 2.373$ is the exponent of fast matrix multiplication; this time complexity is the same as the fastest known time complexity for exact linear regression on a single table. In the case of categorical features (sparse data), using sparse tensors, D^2 can be replaced by a constant that is at least the number of non-zero elements in $\mathbf{J}^T\mathbf{J}$ (which is at least d^2 and at most D^2) using the algorithm in (Abo Khamis et al., 2018).

We state two results with differing leading terms and low-order additive terms, as one may be more useful than the other depending on whether the input tables are dense or sparse.

Theorem 1 (In-Database Subspace Embedding)**.** *Suppose $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{N \times d}$ is a join of two tables, where $\mathbf{T}_1 \in \mathbb{R}^{n_1 \times d_1}, \mathbf{T}_2 \in \mathbb{R}^{n_2 \times d_2}$. Then Algorithm 1 outputs a sketching matrix $\mathbf{S}^* \in \mathbb{R}^{k \times N}$ such that $\tilde{\mathbf{J}} = \mathbf{S}^*\mathbf{J}$ is an ϵ-subspace embedding for $\mathbf{J}$, meaning*

$$\|\mathbf{S}^*\mathbf{J}x\|_2^2 = (1 \pm \epsilon)\|\mathbf{J}x\|_2^2$$

simultaneously for all $x \in \mathbb{R}^d$ with probability[2] *at least $9/10$. The running time to return $\mathbf{S}^*\mathbf{J}$ is the mini-*

[2] We remark that using standard techniques for amplifying the success probability of an SE (see Section 2.3 of (Woodruff et al., 2014)) one can boost the success probability to $1 - \delta$ by repeating the entire algorithm $O(\log \delta^{-1})$ times, increasing the running time by a multiplicative $O(\log \delta^{-1})$ factor. One must then compute the SVD of each of the $O(\log \delta^{-1})$ sketches, which results in an additive $O(kd^{\omega-1} \log \delta^{-1})$ term in the running time, where $\omega \approx 2.373$ is the exponent of matrix multiplication. Note that this additive dependence on d is only slightly ($\approx d^{.373}$) larger than the dependence required for constant probability as stated in the theorem.

mum of $\tilde{O}((n_1 + n_2)d/\epsilon^2 + d^3/\epsilon^2)$ *and* $\tilde{O}((\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2))/\epsilon^2 + (n_1+n_2)/\epsilon^2 + d^5/\epsilon^2)$.[3] *In the former case, we have* $k = \tilde{O}(d^2/\epsilon^2)$*, and in the latter case we have* $k = \tilde{O}(d^4/\epsilon^2)$.

Next, by following a standard reduction from a subspace embedding to an algorithm for regression, we obtain extremely efficient machine precision regression algorithms for two-table database joins.

Theorem 2 (Machine Precision Regression)**.** *Suppose* $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{N\times d}$ *is a join of two tables, where* $\mathbf{T}_1 \in \mathbb{R}^{n_1\times d_1}, \mathbf{T}_2 \in \mathbb{R}^{n_2\times d_2}$*. Let* $\mathbf{U} \subseteq [d]$ *be any subset, and let* $\mathbf{J}_U \in \mathbb{R}^{N\times|U|}$ *be* $\mathbf{J}$ *restricted to the columns in* U*, and let* $b \in \mathbb{R}^N$ *be any column of the join* $\mathbf{J}$*. Then there is an algorithm which outputs* $\hat{x} \in \mathbb{R}^{|U|}$ *such that with probability* $9/10$[4] *we have*

$$\|\mathbf{J}_U\hat{x} - b\|_2 \le (1+\epsilon)\min_{x\in\mathbb{R}^{|U|}} \|\mathbf{J}_U x - b\|_2.$$

The running time required to compute $\hat{x}$ *is the minimum of* $\tilde{O}(((n_1 + n_2)d + d^3)\log(1/\epsilon))$ *and* $\tilde{O}((\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2) + d^5)\log(1/\epsilon))$.

General Joins We next consider arbitrary joins on more than two tables. In this case, we primarily focus on the ridge regression problem $\min_x \|\mathbf{J}x - b\|_2^2 + \lambda\|x\|_2^2$, for a regularization parameter λ. This problem is a popular regularized variant of regression and was considered in the context of database joins in (Abo Khamis et al., 2018). We introduce a general framework to apply sketching methods over arbitrary joins in the supplementary material; our method is able to take a sketch with certain properties as a black box, and can be applied both to TensorSketch (Avron et al., 2014; Pagh, 2013; Pham & Pagh, 2013), as well as recent improvements to this sketch (Ahle et al., 2020; Woodruff & Zandieh, 2020) for certain joins. Unlike previous work, which required computing $\mathbf{J}^T\mathbf{J}$ exactly, we show how to use sketching to approximate this up to high accuracy, where the number of entries of $\mathbf{J}^T\mathbf{J}$ computed depends on the so-called statistical dimension of $\mathbf{J}$, which can be much smaller than the data dimension D.

Evaluation Empirically, we compare our algorithms on various databases to the previous best FAQ-based algorithm of (Abo Khamis et al., 2018), which computes each entry of the covariance matrix $\mathbf{J}^T\mathbf{J}$. For two-table joins, we focus on the standard regression problem. We use the algorithm described in Section 3, replacing the Fast Tensor-Sketch with Tensor-Sketch for better practical performance. For general joins, we focus on the ridge regression problem; such joins can be very high dimensional and ridge regression helps to prevent overfitting. We apply our sketching approach to the entire join and obtain an approximation to it, where our complexity is in terms of the statistical dimension rather than the actual dimension D. Our results demonstrate significant speedups over the previous best algorithm, with only a small sacrifice in accuracy. For example, for the join of two tables in the MovieLens data set, which has 23 features, we obtain a 10-fold speedup over the FAQ-based algorithm, while maintaining a 0.66% relative error. For the natural join of three tables in the real MovieLens data set, which is a join with 24 features, we obtain a 3-fold speedup over the FAQ-based algorithm with only 0.28% MSE relative error. Further details can be found in Section 4.

1.2. Related Work on Sketching Structured Data

The use of specialized sketches for different classes of structured matrices $\mathbf{A}$ has been a topic of substantial interest. The TensorSketch algorithm of (Pagh, 2013) can be applied to Kronecker products $\mathbf{A} = \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_m$ without explicitly computing the product. The efficiency of this algorithm was recently improved by (Ahle et al., 2020). The special case when all $\mathbf{A}_i$ are equal is known as the polynomial kernel, which was considered in (Pham & Pagh, 2013) and extended by (Avron et al., 2014).

Kronecker Product Regression has also been studied for the ℓ_p loss functions (Diao et al., 2017; 2019), which also gave improved algorithms for ℓ_2 regression. In (Avron et al., 2013; Shi & Woodruff, 2019) it is shown that regression on $\mathbf{A}$ can be solved in time $T(\mathbf{A})\cdot\mathtt{poly}(\log(nd))$, where $T(\mathbf{A}) \le \mathtt{nnz}(\mathbf{A})$ is the time needed to compute the matrix-vector product $\mathbf{A}y$ for any $y \in \mathbb{R}^d$. For many classes of $\mathbf{A}$, such as Vandermonde matrices, $T(\mathbf{A})$ is substantially smaller than $\mathtt{nnz}(\mathbf{A})$.

Finally, a flurry of work has used sketching to obtain faster algorithms for low rank approximation of structured matrices. In (Musco & Woodruff, 2017; Bakshi et al., 2019), low rank approximations to positive semidefinite (PSD) matrices are computed in time sublinear in the number of entries of $\mathbf{A}$. This was also shown for distance matrices in (Bakshi & Woodruff, 2018; Indyk et al., 2019; Bakshi et al., 2019).

1.3. Related In-Database Machine Learning Work

The work of (Abo Khamis et al., 2016) introduced Inside-out, a polynomial time algorithm for calculating functional aggregation queries (FAQs) over joins without performing the joins themselves, which can be utilized to train various types of machine learning models. The Inside-Out algorithm builds on several earlier papers, including (Aji & McEliece, 2000; Dechter, 1996; Kohlas & Wilson, 2008; Grohe & Marx, 2006). Relational linear regression, sin-

[3] We use $\tilde{O}$ notation to omit factors of $\log(N)$.

[4] The probability of success here is the same as the probability of success of constructing a subspace embedding; see earlier footnote about amplifying this success probability.

gular value decomposition, and factorization machines are studied extensively in multiple prior works (Rendle, 2013; Kumar et al., 2015b; Schleich et al., 2016; Khamis et al., 2018; Abo Khamis et al., 2018; Kumar et al., 2016; Elgamal et al., 2017; Kumar et al., 2015a). The best known time complexity for training linear regression when the features are continuous, is $O(d^4 m n^{\text{fhtw}} \log(n))$ where fhtw is the fractional hypertree width of the join query. Note that fhtw is 1 for acyclic joins. For categorical features, the time complexity is $O(d^2 m n^{\text{fhtw}+2})$ in the worst-case; however, (Abo Khamis et al., 2018) uses sparse computation of the results to reduce this time depending on the join instance. In the case of polynomial regression, the calculation of pairwise interactions among the features can be time-consuming and it is addressed in (Abo Khamis et al., 2018; Li et al., 2019). Relational support vector machines with Gaussian kernels are studied in (Yang et al.). In (Cheng & Koudas, 2019), a relational algorithm is introduced for Independent Gaussian Mixture Models, which can be used for kernel density estimation by estimating the underlying distribution of the data.

2. Preliminaries

2.1. Database Joins

We first introduce the notion of a *block* of a join, which will be important in our analysis. Let $\mathbf{T}_1, \ldots, \mathbf{T}_m$ be tables, with $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$. Let $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m \in \mathbb{R}^{N \times d}$ be an arbitrary join on the tables $\mathbf{T}_i$. Let Q be the subset of columns which are contained in at least two tables, e.g., the columns which are joined upon. For any subset U of columns and any table T containing a set of columns U', let $T|_U$ be the projection of T onto the columns in $U \cap U'$. Similarly define $r|_U$ for a row r. Let C be the set of columns in $\mathbf{J}$, and let $C_j \subset C$ be the columns contained in T_j. Define the set of *blocks* $\mathcal{B} = \mathcal{B}(\mathbf{J}) \subset \mathbb{R}^{|Q|}$ of the join to be the set of distinct rows in the projection of $\mathbf{J}$ onto Q. In other words, $\mathcal{B}$ is the set of distinct rows which occur in the restriction of $\mathbf{J}$ to the columns being joined on. For any $j \in [m]$, let $\hat{\mathbf{T}}_j \in \mathbb{R}^{n_j \times d}$ be the embedding of the rows of $\mathbf{T}_j$ into the join $\mathbf{J}$, obtained by padding $\mathbf{T}_j$ with zero-valued columns for each column not contained in $\mathbf{T}_j$, and such that for any column c contained in more than one $\mathbf{T}_j$, we define the matrices $\hat{\mathbf{T}}_j$ so that exactly one of the $\hat{\mathbf{T}}_j$ contains c (it does not matter which of the tables containing the column c has c assigned to it in the definition of $\hat{\mathbf{T}}_j$). More formally, we fix any partition $\{\hat{C}_j\}_{j \in [m]}$ of C, such that $C = \cup_j \hat{C}_j$ and $\hat{C}_j \subseteq C_j$ for all j.

For simplicity, given a block $\vec{i} \in \mathcal{B}$, which was defined as a row in $\mathbb{R}^{|Q|}$, we drop the vector notation and write $\vec{i} = i \in \mathcal{B}$. For a given $i = (i_1, \ldots, i_{|Q|}) \in \mathcal{B}$, let $s_{(i)}$ denote the *size* of the block, meaning the number of rows r of the join $\mathbf{J}$ such that i_j is in the j-th column of r for all $j \in Q$. For $i \in \mathcal{B}$, let $\mathbf{T}_j^{(i)}$ be the subset of rows r in $\mathbf{T}_j$ such that $r|_Q = i|_{C_j}$, and similarly define $\hat{\mathbf{T}}_j^{(i)}, \mathbf{J}^{(i)}$ to be the subset of rows r in $\hat{\mathbf{T}}_j$ (respectively $\mathbf{J}$) such that $r|_Q = i|_{\hat{C}_j}$ (respectively $r|_Q = i$). For a row r such that $r|_Q = i$ we say that r "belongs" to the block $i \in \mathcal{B}$. Let $s_{(i),j}$ denote the number of rows of $\mathbf{T}_j^{(i)}$, so that $s_{(i)} = \prod_{j=1}^m s_{(i),j}$.

Figure 1 tables:

T_1

f_1	f_2
1	1
2	1
3	2
4	3
5	4

B_1, B_2

T_2

f_2	f_3
1	1
1	2
2	3
5	4
5	5

$J = T_1 \bowtie T_2$

	f_1	f_2	f_3
$J^{(1)}$	1	1	1
	1	1	2
	2	1	1
	2	1	2
$J^{(2)}$	3	2	3

Figure 1: Example of two table join blocks

$T_1^{(1)}$

f_1	f_2
1	1
2	1

$T_1^{(2)}$

f_1	f_2
3	2

$T_2^{(1)}$

f_2	f_3
1	1
1	2

$T_2^{(2)}$

f_2	f_3
2	3

Figure 2: Examples of $T_i^{(j)}$

As an example, considering the join $T_1(A, B) \bowtie T_2(B, C)$, we have one block for each distinct value of B that is present in both T_1 and T_2, and for a given block $B = b$, the size of the block can be computed as the number of rows in T_1, with $B = b$, multiplied by the number of rows in T_2, with $B = b$.

Using the above notion of blocks of a join, we can construct $\mathbf{J}$ as a stacking of matrices $\mathbf{J}^{(i)}$ for $i \in \mathcal{B}$. For the case of two table joins $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$, we have $\mathbf{J}^{(i)} = \left(\hat{\mathbf{T}}_1^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_2^{(i)}\right) \in \mathbb{R}^{s_{(i)} \times d}$. In other words, $\mathbf{J}^{(i)}$ is the subset of rows of $\mathbf{J}$ contained in block i. Observe that the entire join $\mathbf{J}$ is the result of stacking the matrices $\mathbf{J}^{(i)}$ on top of each other, for all $i \in \mathcal{B}$. In other words, if $\mathcal{B} = \{i_1, i_2, \ldots, i_{|\mathcal{B}|}\}$, the join $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ is given by $\mathbf{J} = \left[(\mathbf{J}^{(i_1)})^T, (\mathbf{J}^{(i_2)})^T, \ldots, (\mathbf{J}^{(i_{|\mathcal{B}|})})^T\right]^T$.

Figure 1 illustrates an example of blocks in a two table join. In this example column f_2 is the column that we are joining the two tables on, and there are two values for f_2 that are present in both tables, namely the values$\{1, 2\}$. Thus $\mathcal{B} = \{B_1, B_2\}$, where $B_1 = 1$ and $B_2 = 2$. In other words, Block B_1 is the block for value 1, and its size is $s_1 = 4$, and similarly B_2 has size $s_2 = 4$. Figure 1 illustrates how the join $\mathcal{J}$ can be written as stacking together the block-matrices $\mathbf{J}^{(1)}$ and $\mathbf{J}^{(2)}$. Figure 2 shows the tables $T_i^{(j)}$ for different values of i and j in the same example.

Finally, for any subset $U \subseteq [N]$, let $\mathbf{J}_U$ denote the set of rows of $\mathbf{J}$ belonging to U. If L is a set of blocks of J, meaning $L \subseteq \mathcal{B}(\mathbf{J})$, then let $\mathbf{J}_L$ denote the set of rows of $\mathbf{J}$ belonging to some block $i \in L$ (recall that a row r "belongs" to a block $i \in L \subseteq \mathcal{B}$ if $r|_Q = i$).

2.2. Linear Algebra

We use boldface font, e.g., $\mathbf{A}, \mathbf{B}, \mathbf{J}$, throughout to denote matrices. Given $\mathbf{A} \in \mathbb{R}^{n \times d}$ with rank r, we write $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ to denote the singular value decomposition (SVD) of $\mathbf{A}$, where $\mathbf{U} \in \mathbb{R}^{n \times r}, \mathbf{V} \in \mathbb{R}^{d \times r}$, and $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$ is a diagonal matrix containing the non-zero singular values of $\mathbf{A}$. For $i \in [d]$, we write $\sigma_i(\mathbf{A})$ to denote the i-th (possibly zero-valued) singular value of $\mathbf{A}$, so that $\sigma_1(\mathbf{A}) \geq \sigma_2(\mathbf{A}) \geq \cdots \geq \sigma_d(\mathbf{A})$. We also use $\sigma_{\max(\mathbf{A})}$ and $\sigma_{\min}(\mathbf{A})$ to denote the maximum and minimum singular values of $\mathbf{A}$ respectively, and let $\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})}$ denote the condition number of $\mathbf{A}$. Let $\mathbf{A}^+$ denote the Moore-Penrose pseudoinverse of $\mathbf{A}$, namely $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T$. Let $\|\mathbf{A}\|_F = (\sum_{i,j} \mathbf{A}_{i,j}^2)^{1/2}$ denote the Frobenius norm of $\mathbf{A}$, and $\|\mathbf{A}\|_2 = \sigma_{\max}(\mathbf{A})$ the spectral norm. We write $\mathbb{I}_n \in \mathbb{R}^{n \times n}$ to denote the n-dimensional identity matrix. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, we write $\texttt{nnz}(\mathbf{A})$ to denote the number of non-zero entries of $\mathbf{A}$. We can assume that each row of the table $\mathbf{T}_j$ is non-zero, since otherwise the row can be removed, and thus $\texttt{nnz}(\mathbf{T}_j) \geq n_i$. Given $\mathbf{A} \in \mathbb{R}^{n \times d}$, we write $\mathbf{A}_{i,*} \in \mathbb{R}^{1 \times d}$ to denote the i-th row (vector) of $\mathbf{A}$, and $\mathbf{A}_{*,i} \in \mathbb{R}^{n \times 1}$ to denote the i-th column (vector) of $\mathbf{A}$.

For values $a, b \in \mathbb{R}$ and $\epsilon > 0$, we write $a = (1 \pm \epsilon)b$ to denote the containment $(1 - \epsilon)b \leq a \leq (1 + \epsilon)b$. For $n \in \mathbb{Z}_+$, let $[n] = \{1, 2, \ldots, n\}$. Throughout, we will use $\tilde{O}(\cdot)$ notation to omit poly$(\log N)$ factors.

Definition 3 (Statistical Dimension). *For a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, and a non-negative scalar λ, the λ-statistical dimension is defined to be $d_\lambda = \sum_i \frac{\lambda_i(\mathbf{A}^T\mathbf{A})}{\lambda_i(\mathbf{A}^T\mathbf{A}) + \lambda}$, where $\lambda_i(\mathbf{A}^T\mathbf{A})$ is the i-th eigenvalue of $\mathbf{A}^T\mathbf{A}$.*

Definition 4 (Subspace Embedding). *For an $\epsilon \geq 0$, we say that $\tilde{\mathbf{A}} \in \mathbb{R}^{m \times d}$ is an ϵ-subspace embedding for $\mathbf{A} \in \mathbb{R}^{n \times d}$ if for all $x \in \mathbb{R}^d$ we have*

$$(1 - \epsilon)\|\mathbf{A}x\|_2 \leq \|\tilde{\mathbf{A}}x\|_2 \leq (1 + \epsilon)\|\mathbf{A}x\|_2.$$

Note that if $\tilde{\mathbf{A}} \in \mathbb{R}^{m \times d}$ is an ϵ-subspace embedding for $\mathbf{A} \in \mathbb{R}^{n \times d}$, in particular this implies that $\sigma_i(\mathbf{A}) = (1 \pm \epsilon)\sigma_i(\tilde{\mathbf{A}})$ for all $i \in [d]$.

Leverage Scores The *leverage score* of the i-th row $\mathbf{A}_{i,*}$ of $\mathbf{A} \in \mathbb{R}^{n \times d}$ is defined to be $\tau_i(\mathbf{A}) = \mathbf{A}_{i,*}(\mathbf{A}^T\mathbf{A})^+\mathbf{A}_{i,*}^T$. Let $\tau(\mathbf{A}) \in \mathbb{R}^n$ be the vector such that $(\tau(\mathbf{A}))_i = \tau_i(\mathbf{A})$. Note that $\tau(\mathbf{A})$ is the diagonal of $\mathbf{A}(\mathbf{A}^T\mathbf{A})^+\mathbf{A}^T$. Our algorithm will utilize *generalized leverage scores*. Given matrices $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{B} \in \mathbb{R}^{n_1 \times d}$, the generalized leverage scores of $\mathbf{A}$ with respect to $\mathbf{B}$ are defined as

$$\tau_i^{\mathbf{B}}(\mathbf{A}) = \begin{cases} \mathbf{A}_{i,*}(\mathbf{B}^T\mathbf{B})^+\mathbf{A}_{i,*}^T & \text{if } \mathbf{A}_{i,*} \perp \ker(\mathbf{B}) \\ 1 & \text{otherwise} \end{cases}$$

Where $\mathbf{A}_{i,*} \perp \ker(\mathbf{B})$ denotes that $\mathbf{A}_{i,*}$ is orthogonal to the kernel of $\mathbf{B}$. Note that for a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ with SVD $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, we have $(\mathbf{B}^T\mathbf{B})^+ = \mathbf{V}\Sigma^{-2}\mathbf{V}^T$. Thus, for any $x \in \mathbb{R}^d$, we have $x^T(\mathbf{B}^T\mathbf{B})^+x = \|x^T\mathbf{V}\Sigma^{-1}\|_2^2$, and in particular $\tau_i^{\mathbf{B}}(\mathbf{A}) = \|\mathbf{A}_{i,*}^T\mathbf{V}\Sigma^{-1}\|_2^2$ if $\mathbf{A}_{i,*} \perp \ker(\mathbf{B})$.

The proof of the following can be found in the supplementary material.

Proposition 1. *If $\mathbf{B}' \in \mathbb{R}^{n_1' \times d}$ is an ϵ-subspace embedding for $\mathbf{B} \in \mathbb{R}^{n_1 \times d}$, and $\mathbf{A} \in \mathbb{R}^{n \times d}$ is any matrix, then $\tau_i^{\mathbf{B}'}(\mathbf{A}) = (1 \pm O(\epsilon))\tau_i^{\mathbf{B}}(\mathbf{A})$*

Our algorithm will employ a mixture of several known oblivious subspace embeddings as tools to construct our overall database join SE. The first result we will need is an improved variant of *Tensor-Sketch*, which is an SE that can be applied quickly to tensor products of matrices.

Lemma 5 (Fast Tensor-Sketch, Theorem 3 of (Ahle et al., 2020)). *Fix any matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_m$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, fix $\epsilon > 0$ and $\lambda \geq 0$. Let $n = n_1 \cdots n_m$ and $d = d_1 \cdots d_m$. Let $\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_m$ have statistical dimension d_λ. Then there is an oblivious randomized sketching algorithm which produces a matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$, where $k = O(d_\lambda m^4/\epsilon^2)$, such that with probability $1 - 1/\texttt{poly}(n)$, we have that for all $x \in \mathbb{R}^d$*

$$\|\mathbf{S}\mathbf{A}x\|_2^2 + \lambda\|x\|_2^2 = (1 \pm \epsilon)(\|\mathbf{A}x\|_2^2 + \lambda\|x\|_2^2).$$

Note for the case of $\lambda = 0$, this implies that $\mathbf{S}\mathbf{A}$ is an ϵ-subspace embedding for $\mathbf{A}$. Moreover, $\mathbf{S}\mathbf{A}$ can be computed in time $\tilde{O}(\sum_{i=1}^m \texttt{nnz}(\mathbf{A}_i)/\epsilon^2 \cdot m^5 + kdm)$.[5]

For the special case of $\lambda = 0$ in Lemma 5, the statistical dimension is d, and Tensor-Sketch is just a standard SE.

Lemma 6 (OSNAP Transform (Nelson & Nguyên, 2013)). *Given any $\mathbf{A} \in \mathbb{R}^{N \times d}$, there is a randomized oblivious sketching algorithm that produces a matrix $\mathbf{W} \in \mathbb{R}^{t \times N}$ with $t = \tilde{O}(d/\epsilon^2)$, such that $\mathbf{W}\mathbf{A}$ can be computed in time $\tilde{O}(\texttt{nnz}(\mathbf{A})/\epsilon^2)$, and such that $\mathbf{W}\mathbf{A}$ is an ϵ-subspace embedding for $\mathbf{A}$ with probability at least $99/100$. Moreover, each column of $\mathbf{W}$ has at most $\tilde{O}(1)$ non-zero entries.*

Lemma 7 (Count-Sketch(Clarkson & Woodruff, 2013)). *For any fixed matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, and any $\epsilon > 0$, there exists an algorithm which produces a matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$, where $k = O(d^2/\epsilon^2)$, such that $\mathbf{S}\mathbf{A}$ is an ϵ-subspace embedding*

[5] Theorem 3 of (Ahle et al., 2020) is written to be applied to the special case of the polynomial kernel, where $A_1 = A_2 = \cdots = A_m$. However, the algorithm itself does not use this fact, nor does it require the factors in the tensor product to be non-distinct.

for $\mathbf{A}$ *with probability at least* $99/100$*. Moreover, each column of* $\mathbf{S}$ *contains exactly one non-zero entry, and therefore* $\mathbf{SA}$ *can be computed in* $O(\mathtt{nnz}(\mathbf{A}))$ *time.*

3. Subspace Embeddings for Two-Table Database Joins

In this section, we will describe our algorithms for fast computation of in-database subspace embeddings for joins of two tables $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$, where $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$, and $\mathbf{J} \in \mathbb{R}^{N \times d}$. As a consequence of our subspace embeddings, we obtain an input sparsity time algorithm for machine precision in-database regression. Here, machine precision refers to a convergence rate of $\log(1/\epsilon)$ to the optimal solution.

Our subspace embedding algorithm can be run with two separate hyper-parameterizations, one of which we refer to as the *dense case* where the tables $\mathbf{T}_1, \mathbf{T}_2$ have many non-zero entries, and the other is referred to as the *sparse case*, where we exploit the sparsity of the tables $\mathbf{T}_1, \mathbf{T}_2$. In the former, we will obtain $\tilde{O}(\frac{1}{\epsilon^2}((n_1 + n_2)d + d^3))$ runtime for construction of our subspace embedding, and in the latter we will obtain $\tilde{O}(\frac{1}{\epsilon^2}(\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2) + d^5))$ time. Thus, when the matrices $\mathbf{T}_1, \mathbf{T}_2$ are dense, we have $(n_1 + n_2)d = \Theta(\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2))$, in which case the former algorithm has a strictly better runtime dependence on n_1, n_2, and d. However, for the many applications where $\mathbf{T}_1, \mathbf{T}_2$ are sparse, the latter algorithm will yield substantial improvements in runtime. By first reading off the sparsity of $\mathbf{T}_1, \mathbf{T}_2$ and choosing the hyperparameters which minimize the runtime, the final runtime of the algorithm is the minimum of the two aforementioned runtimes.

Our main subspace embedding is given in Algorithm 1. The full proofs are deferred to the supplementary material. As noted in Section 2.1, we can describe the join $\mathbf{J}$ as the result of stacking several "blocks" $\mathbf{J}^{(i)}$, where the rows of $\mathbf{J}^{(i)}$ consist of all pairs of concatenations of a row of $\mathbf{T}_1^{(i)}$ and $\mathbf{T}_2^{(i)}$, where the $\mathbf{T}_j^{(i)}$'s partition $\mathbf{T}_j$. We deal separately with blocks i for which $\mathbf{J}^{(i)}$ contains a very large number of rows, and smaller blocks. Formally, we split the set of blocks $\mathcal{B}(\mathbf{J})$ into $\mathcal{B}_{\text{big}}$ and $\mathcal{B}_{\text{small}}$. For each block $\mathbf{J}^{(i)}$ from $\mathcal{B}_{\text{big}}$, we apply a fast tensor sketch transform to obtain a subspace embedding for that block.

For the smaller blocks, however, we need a much more involved routine. Our algorithm computes a random sample of the rows of the blocks $\mathbf{J}^{(i)}$ from $\mathcal{B}_{\text{small}}$, denoted $\tilde{\mathbf{J}}_{\text{small}}$. Using the results of (Cohen et al., 2015), it follows that sampling sufficiently many rows from the distribution induced by the generalized leverage scores $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ of $\mathbf{J}_{\text{small}}$ with respect to $\tilde{\mathbf{J}}_{\text{small}}$ yields a subspace embedding of $\mathbf{J}_{\text{small}}$. However, it is not possible to write down (let alone compute) all the values $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$, since there can be more rows i in $\mathbf{J}_{\text{small}}$ than our entire allowable running time.

Algorithm 1 Subspace embedding for join $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$.

1: In the *dense* case, set $\gamma = 1$. In the *sparse* case, set $\gamma = d$. Compute block sizes $s_{(i)}$, and let $\mathcal{B}_{\text{big}} = \{i \in \mathcal{B} \mid \max_j\{s_{(i),j}\} \geq d \cdot \gamma\}$. Set $\mathcal{B}_{\text{small}} = \mathcal{B} \setminus \mathcal{B}_{\text{big}}$, $n_{\text{small}} = |\mathcal{B}_{\text{small}}|$.

2: For each $i \in \mathcal{B}_{\text{big}}$, generate a Fast Tensor-Sketch matrix $\mathbf{S}^i \in \mathbb{R}^{t \times s_{(i)}}$ (Lemma 5) and compute $\mathbf{S}^i\mathbf{J}^{(i)}$.

3: Let $\tilde{\mathbf{J}}_{\text{big}}$ be the matrix from stacking the matrices $\{\mathbf{S}^i\mathbf{J}^{(i)}\}_{i \in \mathcal{B}_{\text{big}}}$. Generate a Count-Sketch matrix $\mathbf{S}'$ (Lemma 7) and compute $\mathbf{S}'\tilde{\mathbf{J}}_{\text{big}}$.

4: Let $\mathbf{J}_{\text{small}} = \mathbf{J}_{\mathcal{B}_{\text{small}}}$, and sample uniformly a subset U of $m = \Theta((n_1 + n_2)/\gamma)$ rows of $\mathbf{J}_{\mathcal{B}_{small}} \in \mathbb{R}^{n_{\text{small}} \times d}$ and form the matrix $\tilde{\mathbf{J}}_{\text{small}} = (\mathbf{J}_{\text{small}})_U \in \mathbb{R}^{m \times d}$.

5: Generate OSNAP transform $\mathbf{W}$ (Lemma 6), compute $\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}$ and the SVD $\mathbf{W}\tilde{\mathbf{J}}_{\text{small}} = \mathbf{U}\Sigma\mathbf{V}^T$.

6: Generate Gaussian matrix $\mathbf{G} \in \mathbb{R}^{d \times t}$ with entries drawn i.i.d. from $\mathcal{N}(0, 1/t^2)$, $t = \Theta(\log N)$, and Gaussian vector $g \sim \mathcal{N}(0, \mathbb{I}_d)$.

7: For all rows i of $\mathbf{J}_{\text{small}}$, set $\|(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g\|_2^2 = \alpha_i$, and

$$\tilde{\tau}_i = \begin{cases} 1 & \text{if } \alpha_i > 0 \\ \|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\Sigma^{-1}\mathbf{G}\|_2^2 & \text{otherwise} \end{cases}.$$

8: Using Algorithms 2 and 3 with $\mathbf{Y} = \mathbf{V}\Sigma^{-1}\mathbf{G}$, construct diagonal row sampling matrix $\mathbf{S} \in \mathbb{R}^{n_{\text{small}} \times n_{\text{small}}}$ such that $\mathbf{S}_{i,i} = \frac{1}{\sqrt{p_i}}$ with probability p_i, and $\mathbf{S}_{i,i} = 0$ otherwise, where $p_i \geq \min\left\{1, \frac{\log d}{\epsilon^2} \cdot \tilde{\tau}_i\right\}$

9: Return $\tilde{\mathbf{J}}$, where $\tilde{\mathbf{J}}$ is the result of stacking the matrices $\mathbf{S}'\tilde{\mathbf{J}}_{\text{big}}$ with $\mathbf{S}\mathbf{J}_{\text{small}}$.

To handle this issue, we first note that by Proposition 1 and the discussion prior to it, the value $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ is well-approximated by $\|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\Sigma^{-1}\|_2^2$, which in turn is well-approximated by $\|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\Sigma^{-1}\mathbf{G}\|_2^2$ if $\mathbf{G}$ is a Gaussian matrix with only a small $\Theta(\log N)$ number of columns. Thus, sampling from the generalized leverage scores $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ can be approximately reduced to the problem of sampling a row i from $\mathbf{J}_{\text{small}}\mathbf{Y}$ with probability proportional to $\|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{Y}\|_2^2$, where $\mathbf{Y}$ is any matrix given as input. We then design a fast algorithm which accomplishes precisely this task: namely, for any join $\mathbf{J}'$ and input matrix $\mathbf{Y}$ with a small number of columns, it samples rows from $\mathbf{J}'\mathbf{Y}$ with probability proportional to the squared row norms of $\mathbf{J}'\mathbf{Y}$. Since $\mathbf{J}_{\text{small}} = \mathbf{J}'$ is itself a database join, this is the desired sampler. This procedure is given in Algorithms 2 (pre-processing step) and 3 (sampling step). We can apply this sampling primitive to efficiently sample from the generalized leverage scores in time substantially

Algorithm 2 Pre-processing step for fast ℓ_2-sampling of rows of $\mathbf{J} \cdot \mathbf{Y}$, given input $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ and $\mathbf{Y} \in \mathbb{R}^{d \times r}$.

1: For each block $i \in \mathcal{B}$, compute $a_{(i)} = \hat{T}_1^{(i)} \mathbf{Y}$ and $b_{(i)} = \hat{T}_2^{(i)}$
2: For each block $i \in \mathcal{B}$, construct a binary tree $\tau_{(i)}(\mathbf{T}_1), \tau_{(i)}(\mathbf{T}_2)$ as follows:
3: Each node $v \in \tau_{(i)}(\mathbf{T}_j)$ is a vector $v \in \mathbb{R}^{3r}$
4: If v is not a leaf, then $v = v_{\text{lchild}} + v_{\text{rchild}}$ with $v_{\text{lchild}}, v_{\text{rchild}}$ the left and right children of v.
5: The leaves of $\tau_{(i)}(\mathbf{T}_j)$ are given by the set $\{v_l^{(i),j} = (v_{l,1}^{(i),j}, v_{l,2}^{(i),j}, \ldots, v_{l,r}^{(i),j}) \in \mathbb{R}^{3r} \mid l \in [s_{(i),j}]\}$, where $v_{l,q}^{(i),1} = (1, 2(a_{(i)})_{l,q}, (a_{(i)})_{l,q}^2) \in \mathbb{R}^3$ and $v_{l,q}^{(i),2} = ((b_{(i)})_{l,q}^2, (b_{(i)})_{l,q}, 1) \in \mathbb{R}^3$.
6: Compute the values $\langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2)) \rangle$ for all $i \in \mathcal{B}$, and also compute the sum $\sum_i \langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2)) \rangle$.

Algorithm 3 Sampling step for fast ℓ_2-sampling of rows of $\mathbf{J} \cdot \mathbf{Y}$.

1: Sample a block $i \in \mathcal{B}$ with probability proportional to $\langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2)) \rangle$.
2: Sample $l_1 \in [s_{(i),1}]$ with probability proportional to $\langle v_{l_1}^{(i),1}, \text{root}(\tau_i(\mathbf{T}_2)) \rangle$.
3: Sample $l_2 \in [s_{(i),2}]$ with probability proportional to $\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2} \rangle$.
4: **Return** row corresponding to (l_1, l_2) in block i.

less than constructing $\mathbf{J}_{\text{small}}$, which ultimately allows for our final subspace embedding guarantee of Theorem 1.

Finally, to obtain our input sparsity runtime machine precision regression algorithm, we apply our subspace embedding with constant ϵ to precondition the join $\mathbf{J}$, after which the regression problem can be solved quickly via gradient descent. While a general gradient step is not always possible to compute efficiently with respect to the join $\mathbf{J}$, we demonstrate that when the products used in the gradient step arise from vectors in the column span of $\mathbf{J}$, the updates can be computed efficiently, which will yield our main regression result (Theorem 2). The full proofs are deferred to the supplementary material.

4. Evaluation

We study the performance of our sketching method on several real datasets, both for two-table joins and general joins.[6] We first introduce the datasets we use in the experiments. We consider two datasets: LastFM (Cantador et al., 2011) and MovieLens (Harper & Konstan, 2015). Both of them contain several relational tables. We will compare our algorithm with the FAQ-based algorithm on the joins of some relations.

The LastFM dataset has three relations: **Userfriends** (the friend relations between users), **Userartists** (the artists listened by each user) and **Usertaggedartiststimestamps** (the tag assignments of artists provided by each particular user along with the timestamps).

The MovieLens dataset also has three relations: **Ratings** (the ratings of movies given by the users and the timestamps), **Users** (gender, age, occupation, and zip code information of each user), **Movies** (release year and the category of each movie).

4.1. Two-Table Joins

In the experiments for two-table joins, we solve the regression problem $\min_x \|\mathbf{J}_U x - b\|_2^2$, where $\mathbf{J} = T_1 \bowtie T_2 \in \mathbb{R}^{N \times d}$ is a join of two tables, $U \subset [d]$ and b is one of the columns of $\mathbf{J}$. In our experiments, suppose column p is the column we want to predict. We will set $U = [d] \setminus \{p\}$ and b to be the p-th column of $\mathbf{J}$.

To solve the regression problem, the FAQ-based algorithm computes the covariance matrix $\mathbf{J}^T\mathbf{J}$ by running the FAQ algorithm for every two columns, and then solves the normal equations $\mathbf{J}^T\mathbf{J}x = \mathbf{J}^T b$. Our algorithm will compute a subspace embedding $\tilde{\mathbf{J}}$, and then solve the regression problem $\min_x \|\tilde{\mathbf{J}}_U x - \tilde{b}\|_2^2$, i.e., solve $\tilde{\mathbf{J}}^T\tilde{\mathbf{J}}x = \tilde{\mathbf{J}}^T\tilde{b}$.

We compare our algorithm to the FAQ-based algorithm on the LastFM and MovieLens datasets. The FAQ-based algorithm employs the FAQ-based algorithm to calculate each entry in $\mathbf{J}^T\mathbf{J}$.

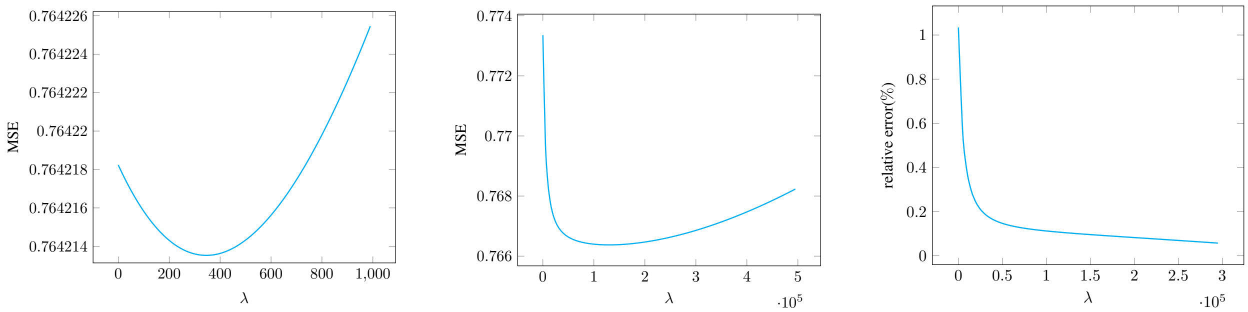
For the LastFM dataset, we consider the join of **Userartists** and **Usertaggedartiststimestamps**: $\mathbf{J}_1 = \mathbf{UA} \bowtie_{\mathbf{UA}.\text{user}=\mathbf{UTA}.\text{user}} \mathbf{UTA}$. Our regression task is to predict how often a user listens to an artist based on the tags. For the MovieLens dataset, we consider the join of **Ratings** and **Movies**: $\mathbf{J}_2 = \mathbf{R} \bowtie_{\mathbf{R}.\text{movie}=\mathbf{M}.\text{movie}} \mathbf{M}$. Our regression task is to predict the rating that a user gives to a movie.

In our experiments, we do the dataset preparation mentioned in (Schleich et al., 2016), to normalize the values in each column to range $[0, 1]$. For each column, let $v_{\max}$ and $v_{\min}$ denote the maximum value and minimum value in this column. We normalize each value v to $(v - v_{\min})/(v_{\max} - v_{\min})$.

4.2. General Joins

For general joins, we consider the ridge regression problem. Specifically, our goal is to find a vector x that minimizes $\|\mathbf{J}x - b\|_2^2 + \lambda\|x\|_2^2$, where $\mathbf{J} = \mathbf{T}_1 \bowtie \cdots \bowtie \mathbf{T}_m \in \mathbb{R}^{N \times d}$ is an arbitrary join, b is one of the columns of $\mathbf{J}$ and $\lambda > 0$ is the regularization parameter. The optimal solution to the ridge regression problem can be found by solving the

[6] Code available at https://github.com/AnonymousFireman/ICML_code



(a) MSE vs. λ for the FAQ-based algorithm (b) MSE vs. λ for our algorithm (c) Relative Error vs. λ

Figure 3: Experimental Results for General Joins

Table 1: Experimental Results for Two-Table Joins

	n_1	n_2	d	T_{bf}	T_{ours}	err
$\mathbf{J}_1$	92834	186479	6	.034	.011	0.70%
$\mathbf{J}_2$	1000209	3883	23	.820	.088	0.66%

normal equations $(\mathbf{J}^T\mathbf{J} + \lambda \mathbb{I}_d)x = \mathbf{J}^T b$.

The FAQ-based algorithm is the same as in the experiment for two-table joins. It directly runs the FAQ algorithm a total of $d(d+1)/2$ times to compute every entry of $\mathbf{J}^T\mathbf{J}$.

We run our algorithm, discussed in the supplementary, and the FAQ-based algorithm on the MovieLens-25m dataset, which is the largest of the MovieLens(Harper & Konstan, 2015) datasets. We consider the join of **Ratings**, **Users** and **Movies**: $\mathbf{J}_3 = \mathbf{R} \bowtie_{\mathbf{R}.\text{user}=\mathbf{U}.\text{user}} \mathbf{U} \bowtie_{\mathbf{U}.\text{movie}=\mathbf{M}.\text{movie}} \mathbf{M}$. Our regression task is to predict the rating that a user gives to a movie.

4.3. Results

We run the FAQ-based algorithm and our algorithm on those joins and compare their running times. To measure accuracy, we compute the relative mean-squared error, given by:

$$\text{err} = \frac{\|\mathbf{J}_U x_{\text{ours}} - b\|_2^2 - \|\mathbf{J}_U x_{\text{bf}} - b\|_2^2}{\|\mathbf{J}_U x_{\text{bf}} - b\|_2^2}$$

in the experiments for two-table joins, where x_{bf} is the solution given by the FAQ-based algorithm, and x_{ours} is the solution given by our algorithm. All results (runtime, accuracy) are averaged over 5 runs of each algorithm.

In our implementation, we adjust the target dimension in our sketching algorithm for each experiment, as in practice it appears unnecessary to parameterize according to the worst-case theoretical bounds when the number of features is small, as in our experiments. Additionally, for two-table joins, we replace the Fast Tensor-Sketch with Tensor-Sketch ((Ahle et al., 2020; Pagh, 2013)) for the same reason. The implementation is written in MATLAB and run on an Intel Core i7-7500U CPU with 8GB of memory.

We let T_{bf} be the running time of the FAQ-based algorithm and T_{ours} be the running time of our approach, measured in seconds. Table 1 shows the results of our experiments for two-table joins. From that we can see our approach can give a solution with relative error less than 1%, and its running time is significantly less than that of the FAQ-based algorithm.

For general joins, due to the size of the dataset, we implement our algorithm in Taichi (Hu et al., 2019; 2020) and run it on an Nvidia GTX1080Ti GPU. We split the dataset into a training set and a validation set, run the regression on the training set and measure the MSE (mean squared error) on the validation set. We fix the target dimension and try different values of λ to see which value achieves the best MSE.

Our algorithm runs in 0.303s while the FAQ-based algorithm runs in 0.988s. The relative error of MSE (namely, $\frac{\text{MSE}_{\text{ours}} - \text{MSE}_{\text{bf}}}{\text{MSE}_{\text{bf}}}$, both measured under the optimal λ) is only 0.28%.

We plot the MSE vs. λ curve for the FAQ-based algorithm and our algorithm in Figure 3(b) and 3(a). We observe that the optimal choice of λ is much larger in the sketched problem than in the original problem. This is because the statistical dimension d_λ decreases as λ increases. Since we fix the target dimension, ϵ thus decreases. So a larger λ can give a better approximate solution, yielding a better MSE even if it is not the best choice in the unsketched problem.

We also plot the relative error of the objective function in Figure 3(c). For ridge regression it becomes

$$\frac{\left(\|\mathbf{J}x_{\text{ours}} - b\|_2^2 + \lambda\|x_{\text{ours}}\|_2^2\right) - \left(\|\mathbf{J}x_{\text{bf}} - b\|_2^2 + \lambda\|x_{\text{bf}}\|_2^2\right)}{\|\mathbf{J}x_{\text{bf}} - b\|_2^2 + \lambda\|x_{\text{bf}}\|_2^2}.$$

We can see that the relative error decreases as λ increases in accordance with our theoretical analysis.

5. Conclusion

In this work, we demonstrate that subspace embeddings for database join queries can be computed in time substantially faster than forming the join, yielding input sparsity time algorithms for regression on joins of two tables up to machine precision, and we extend our results to ridge regression on arbitrary joins. Our results improve on the state-of-the-art FAQ-based algorithms for performing in-database regression on joins. Empirically, our algorithms are substantially faster than the state-of-the-art algorithms for this problem.

Acknowledgments: The authors would like to thank support from the National Science Foundation (NSF) grant No. CCF-1815840, the Office of Naval Research (ONR) grant N00014-18-1-256, and a Simons Investigator Award.

References

https://cloud.google.com/bigquery-ml/docs/bigqueryml-intro.

https://www.relational.ai/.

Abo Khamis, M., Ngo, H. Q., and Rudra, A. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 13–28. ACM, 2016.

Abo Khamis, M., Ngo, H. Q., Nguyen, X., Olteanu, D., and Schleich, M. In-database learning with sparse tensors. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, SIGMOD/PODS '18, pp. 325–340, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-4706-8. doi: 10.1145/3196959.3196960. URL http://doi.acm.org/10.1145/3196959.3196960.

Ahle, T. D. and Knudsen, J. B. Almost optimal tensor sketch. *arXiv preprint arXiv:1909.01821*, 2019.

Ahle, T. D., Kapralov, M., Knudsen, J. B., Pagh, R., Velingker, A., Woodruff, D. P., and Zandieh, A. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 141–160. SIAM, 2020.

Aji, S. M. and McEliece, R. J. The generalized distributive law. *IEEE transactions on Information Theory*, 46(2): 325–343, 2000.

Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

Alon, N., Gibbons, P. B., Matias, Y., and Szegedy, M. Tracking join and self-join sizes in limited storage. *J. Comput. Syst. Sci.*, 64(3):719–747, 2002. doi: 10.1006/jcss.2001.1813. URL https://doi.org/10.1006/jcss.2001.1813.

Avron, H., Sindhwani, V., and Woodruff, D. P. Sketching structured matrices for faster nonlinear regression. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 2994–3002, 2013.

Avron, H., Nguyen, H., and Woodruff, D. Subspace embeddings for the polynomial kernel. In *Advances in neural information processing systems*, pp. 2258–2266, 2014.

Bakshi, A. and Woodruff, D. Sublinear time low-rank approximation of distance matrices. In *Advances in Neural Information Processing Systems*, pp. 3782–3792, 2018.

Bakshi, A., Chepurko, N., and Woodruff, D. P. Robust and sample optimal algorithms for psd low-rank approximation. *ArXiv*, abs/1912.04177, 2019.

Cantador, I., Brusilovsky, P., and Kuflik, T. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM.

Cheng, Z. and Koudas, N. Nonlinear models over normalized data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1574–1577. IEEE, 2019.

Clarkson, K. L. and Woodruff, D. P. Low rank approximation and regression in input sparsity time. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pp. 81–90, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320290. doi: 10.1145/2488608.2488620. URL https://doi.org/10.1145/2488608.2488620.

Cohen, M. B., Lee, Y. T., Musco, C., Musco, C., Peng, R., and Sidford, A. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pp. 181–190, 2015.

Dechter, R. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, 1996.

Demmel, J., Dumitriu, I., and Holtz, O. Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91, 2007.

Diao, H., Song, Z., Sun, W., and Woodruff, D. P. Sketching for kronecker product regression and p-splines. *CoRR*, abs/1712.09473, 2017. URL http://arxiv.org/abs/1712.09473.

Diao, H., Jayaram, R., Song, Z., Sun, W., and Woodruff, D. P. Optimal sketching for kronecker product regression and low rank approximation, 2019.

Elgamal, T., Luo, S., Boehm, M., Evfimievski, A. V., Tatikonda, S., Reinwald, B., and Sen, P. Spoof: Sum-product optimization and operator fusion for large-scale machine learning. In *CIDR*, 2017.

Grohe, M. and Marx, D. Constraint solving via fractional edge covers. In *SODA*, pp. 289–298, 2006.

Gucht, D. V., Williams, R., Woodruff, D. P., and Zhang, Q. The communication complexity of distributed set-joins with applications to matrix multiplication. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pp. 199–212, 2015.

Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL https://doi.org/10.1145/2827872.

Hellerstein, J., Ré, C., Schoppmann, F., Wang, D. Z., Fratkin, E., Gorajek, A., Ng, K. S., Welton, C., Feng, X., Li, K., et al. The madlib analytics library or mad skills, the sql. *arXiv preprint arXiv:1208.4165*, 2012.

Hu, Y., Li, T.-M., Anderson, L., Ragan-Kelley, J., and Durand, F. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019.

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., and Durand, F. Difftaichi: Differentiable programming for physical simulation. *ICLR*, 2020.

Indyk, P., Vakilian, A., Wagner, T., and Woodruff, D. Sample-optimal low-rank approximation of distance matrices. *arXiv preprint arXiv:1906.00339*, 2019.

Khamis, M. A., Ngo, H. Q., Nguyen, X., Olteanu, D., and Schleich, M. Ac/dc: in-database learning thunderstruck. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, pp. 8. ACM, 2018.

Kohlas, J. and Wilson, N. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.

Kumar, A., Jalal, M., Yan, B., Naughton, J., and Patel, J. M. Demonstration of santoku: optimizing machine learning over normalized data. *Proceedings of the VLDB Endowment*, 8(12):1864–1867, 2015a.

Kumar, A., Naughton, J., and Patel, J. M. Learning generalized linear models over normalized data. In *ACM SIGMOD International Conference on Management of Data*, pp. 1969–1984, 2015b.

Kumar, A., Naughton, J., Patel, J. M., and Zhu, X. To join or not to join?: Thinking twice about joins before feature selection. In *International Conference on Management of Data*, pp. 19–34, 2016.

Li, M., Miller, G. L., and Peng, R. Iterative row sampling. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pp. 127–136. IEEE, 2013.

Li, S., Chen, L., and Kumar, A. Enabling and optimizing non-linear feature interactions in factorized linear algebra. In *Proceedings of the 2019 International Conference on Management of Data*, pp. 1571–1588, 2019.

Musco, C. and Woodruff, D. P. Sublinear time low-rank approximation of positive semidefinite matrices. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 672–683. IEEE, 2017.

Nelson, J. and Nguyên, H. L. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 ieee 54th annual symposium on foundations of computer science*, pp. 117–126. IEEE, 2013.

Pagh, R. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)*, 5(3):1–17, 2013.

Pham, N. and Pagh, R. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 239–247, 2013.

Rendle, S. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6, pp. 337–348. VLDB Endowment, 2013.

Schleich, M., Olteanu, D., and Ciucanu, R. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pp. 3–18. ACM, 2016. ISBN 978-1-4503-3531-7.

Shi, X. and Woodruff, D. P. Sublinear time numerical linear algebra for structured matrices. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pp. 4918–4925, 2019. doi: 10.1609/aaai.v33i01.33014918. URL https://doi.org/10.1609/aaai.v33i01.33014918.

Van Loan, C. F. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1-2):85–100, 2000.

Woodruff, D. P. and Zandieh, A. Near input sparsity time kernel embeddings via adaptive sampling. In *International Conference on Machine Learning (ICML)*, 2020.

Woodruff, D. P. et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

Yang, K., Gao, Y., Liang, L., Yao, B., Wen, S., and Chen, G. Towards factorized svm with gaussian kernels over normalized data.

6. Additional Preliminaries

We now present various additional preliminaries needed for our main result on in-database subspace embeddings. We begin with a table of notation for database joins and the *blocks* of a join, as introduced in Section 2.1.

Table 2: Table of Notation

T_i	$\triangleq$	a $n_i \times d_i$ sized table
$\mathbf{J}$	$\triangleq$	join of all tables, i.e., $\mathbf{J} = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$
C	$\triangleq$	set of columns of $\mathbf{J}$
C_i	$\triangleq$	set of columns of T_i
$\hat{C}_i$	$\triangleq$	Partition of C such that $\hat{C}_i \subseteq C_i$ for $i \in [m]$.
$T\|_U$	$\triangleq$	projection of T onto columns $U \cap U'$, where U' are the columns of T
$\hat{T}_i$	$\triangleq$	Result of padding $T_i\|_{\hat{C}_i}$ with zero-valued columns in $C \setminus \hat{C}_i$
$\mathcal{B}$	$\triangleq$	set of *blocks*, i.e., distinct rows of $\mathbf{J}\|_Q$
$s_{(i)}$	$\triangleq$	size of block i, i.e., number of rows r of $\mathbf{J}$ with $r\|_q = i \in \mathcal{B}$
$T_j^{(i)}$	$\triangleq$	subset of rows r in $\mathbf{T}_j$ such that $r\|_Q = i\|_{C_j}$
$\hat{T}_j^{(i)}$	$\triangleq$	subset of rows r in $\hat{T}_j$ such that $r\|_Q = i\|_{\hat{C}_j}$
$\mathbf{J}_j^{(i)}$	$\triangleq$	subset of rows r in $\mathbf{J}$ such that $r\|_Q = i$

6.1. Leverage Scores

We begin with an extended discussion of leverage scores. Recall that the leverage score of the i-th row $\mathbf{A}_{i,*}$ of $\mathbf{A} \in \mathbb{R}^{n \times d}$ is defined to be $\tau_i(\mathbf{A}) = \mathbf{A}_{i,*}(\mathbf{A}^T\mathbf{A})^+\mathbf{A}_{i,*}^T$. Let $\tau(\mathbf{A}) \in \mathbb{R}^n$ be the vector such that $(\tau(\mathbf{A}))_i = \tau_i(\mathbf{A})$. Then $\tau(\mathbf{A})$ is the diagonal of $\mathbf{A}(\mathbf{A}^T\mathbf{A})^+\mathbf{A}^T$, which is a projection matrix. Thus $\tau_i(\mathbf{A}) \leq 1$ for all $i \in [n]$. It is also easy to see that $\sum_{i=1}^n \tau_i(\mathbf{A}) \leq d$ (Cohen et al., 2015). Our algorithm will utilize the *generalized leverage scores*. Given matrices $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{B} \in \mathbb{R}^{n_1 \times d}$, the generalized leverage scores of $\mathbf{A}$ with respect to $\mathbf{B}$ are defined as

$$\tau_i^{\mathbf{B}}(\mathbf{A}) = \begin{cases} \mathbf{A}_{i,*}(\mathbf{B}^T\mathbf{B})^+\mathbf{A}_{i,*}^T & \text{if } \mathbf{A}_{i,*} \perp \ker(\mathbf{B}) \\ 1 & \text{otherwise} \end{cases}$$

We remark that in the case were $\mathbf{A}_{i,*}$ has a component in the kernel (null space) of B, denoted by $\ker(\mathbf{B})$, $\tau_i^{\mathbf{B}}(\mathbf{A})$ is defined to be ∞ in (Cohen et al., 2015). However, as stated in that paper, this definition was simply for notational convenience, and the results would equivalently hold setting $\tau_i^{\mathbf{B}}(\mathbf{A}) = 1$ in this case. Note that for a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ with SVD $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, we have $(\mathbf{B}^T\mathbf{B})^+ = \mathbf{V}\Sigma^{-2}\mathbf{V}^T$. Thus, for any $x \in \mathbb{R}^d$, we have $x^T(\mathbf{B}^T\mathbf{B})^+x = \|x^T\mathbf{V}\Sigma^{-1}\|_2^2$, and in particular $\tau_i^{\mathbf{B}}(\mathbf{A}) = \|\mathbf{A}_{i,*}^T\mathbf{V}\Sigma^{-1}\|_2^2$ if

$\mathbf{A}_{i,*} \perp \ker(\mathbf{B})$, where $\mathbf{A}_{i,*} \perp \ker(\mathbf{B})$ means $\mathbf{A}_{i,*}$ is perpendicular to the kernel of $\mathbf{B}$.

Proposition 2. *If $\mathbf{B}' \in \mathbb{R}^{n_1' \times d}$ is an ϵ-subspace embedding for $\mathbf{B} \in \mathbb{R}^{n_1 \times d}$, and $\mathbf{A} \in \mathbb{R}^{n \times d}$ is any matrix, then $\tau_i^{\mathbf{B}'}(\mathbf{A}) = (1 \pm O(\epsilon))\tau_i^{\mathbf{B}}(\mathbf{A})$*

Proof. If $\mathbf{B}'$ is an ϵ-subspace embedding for $\mathbf{B}$, then the spectrum of $(\mathbf{B}'(\mathbf{B}')^T)^+$ is a $(1 \pm \epsilon)^{-2}$ approximation to the spectrum of $(\mathbf{B}\mathbf{B}^T)^+$, so $x^T(\mathbf{B}'(\mathbf{B}')^T)^+x = (1 \pm \epsilon)^{-2} x^T(\mathbf{B}\mathbf{B}^T)^+x$ for all $x \in \mathbb{R}^d$, which completes the proof. □

7. Subspace Embeddings for Two-Table Database Joins

In this section, we formally describe and prove the correctness of our algorithm for the fast computation of in-database subspace embeddings for the join of two tables $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$, where $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$, and $\mathbf{J} \in \mathbb{R}^{N \times d}$. These algorithms yield input-sparsity runtime solvers for in-database machine precision regression. Our full algorithm is given formally below in Algorithm 4. It uses two crucial subroutines, Algorithms 5 and 6, which efficiently sample rows of a portion $\mathbf{J}_{\text{small}}$ of the design matrix $\mathbf{J}$. Note that these three aforementioned algorithms are precisely the same as Algorithms 1, 2, and 3 (respectively), but are reproduced here in the supplementary for convenience.

We will begin by proving our main technical sampling result, which proceeds in a series of lemmas, and demonstrates that the construction of the diagonal sampling matrix $\mathbf{S}_{\text{small}}$ in Algorithm 4 can be carried out extremely quickly.

Proposition 3. *Let $\mathbf{J}_{\text{small}} \in \mathbb{R}^{n_{\text{small}} \times d}$ be the matrix constructed as in Algorithm 4. Then we have $n_{\text{small}} \leq (n_1 + n_2)d \cdot \gamma$, where $\gamma = 1$ in the dense case and $\gamma = d$ in the sparse case.*

Proof. Recall that $\mathbf{J}_{\text{small}}$ consists of all blocks i of $\mathbf{J}$ with $\max\{s_{(i),1}, s_{(i),2}\} < d \cdot \gamma$, and thus $s_{(i)} \leq d^2\gamma^2$. The total number n_{small} of rows in $\mathbf{J}_{\text{small}}$ is then $\sum_{i \in \mathcal{B}_{\text{small}}} s_{(i),1} \cdot s_{(i),2} \leq \|s^1_{\text{small}}\|_2 \|s^2_{\text{small}}\|_2$ by the Cauchy-Schwarz inequality, where s^j_{small} is the vector with coordinates given by the values $s_{(i),j}$ for $i \in \mathcal{B}_{\text{small}}$. Observe that these vectors admit the ℓ_1 bound of $\|s^j_{\text{small}}\|_1 \leq n_j$ since each table $\mathbf{T}_j$ has only n_j rows. Moreover, they admit the ℓ_∞ bound of $\|s^j_{\text{small}}\|_\infty \leq d\gamma$. With these two constraints, it is standard that the ℓ_2 norm is maximized by placing all of the ℓ_1 mass on coordinates with value given by the ℓ_∞ bound. It follows that $\|s^j_{\text{small}}\|_2$ is maximized by having $n_i/(d\gamma)$ coordinates equal to $d\gamma$, giving $\|s^j_{\text{small}}\|_2^2 \leq n_j d\gamma$ for $j \in [2]$, so $n_{\text{small}} \leq \|s^1_{\text{small}}\|_2 \|s^2_{\text{small}}\|_2 \leq (n_1 + n_2)d\gamma$ as required. □

We now demonstrate how we can quickly ℓ_2 sample rows from a join-vector or join-matrix product after input sparsity time pre-processing. This procedure is split into two algorithms, Algorithm 5 and 6. Algorithm 5 is an input sparsity time pre-processing step, which given $\mathbf{J} \in \mathbb{R}^{n \times d}$ and $\mathbf{Y} \in \mathbb{R}^{d \times t}$, constructs several binary tree data structures. Algorithm 6 then uses these data structures to sample a row of the product $\mathbf{JY}$ with probability proportional to its ℓ_2 norm, in time $O(\log N)$.

Algorithm 4 Subspace embedding for join $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$.

1: In the *dense* case set $\gamma = 1$. In the *sparse* case, case $\gamma = d$. Compute block sizes $s_{(i)}$, and let $\mathcal{B}_{\text{big}} = \{i \in \mathcal{B} \mid \max_j\{s_{(i),j}\} \geq d \cdot \gamma\}$. Set $\mathcal{B}_{\text{small}} = \mathcal{B} \setminus \mathcal{B}_{\text{big}}$, $n_{\text{small}} = |\mathcal{B}_{\text{small}}|$.
2: For each $i \in \mathcal{B}_{\text{big}}$, generate a Fast Tensor-Sketch matrix $\mathbf{S}^i \in \mathbb{R}^{t \times s_{(i)}}$ (Lemma 5) and compute $\mathbf{S}^i\mathbf{J}^{(i)}$.
3: Let $\tilde{\mathbf{J}}_{\text{big}}$ be the matrix from stacking the matrices $\{\mathbf{S}^i\mathbf{J}^{(i)}\}_{i \in \mathcal{B}_{\text{big}}}$. Generate a Count-Sketch matrix $\mathbf{S}'$ (Lemma 7) and compute $\mathbf{S}'\tilde{\mathbf{J}}_{\text{big}}$.
4: Let $\mathbf{J}_{\text{small}} = \mathbf{J}_{\mathcal{B}_{\text{small}}}$, and sample uniformly a subset U of $m = \Theta((n_1 + n_2)/\gamma)$ rows of $\mathbf{J}_{\mathcal{B}_{small}} \in \mathbb{R}^{n_{\text{small}} \times d}$ and form the matrix $\tilde{\mathbf{J}}_{\text{small}} = (\mathbf{J}_{\text{small}})_U \in \mathbb{R}^{m \times d}$.
5: Generate OSNAP transform $\mathbf{W}$ (Lemma 6) and compute $\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}$ and the SVD $\mathbf{W} \cdot \tilde{\mathbf{J}}_{\text{small}} = \mathbf{U}\Sigma\mathbf{V}^T$.
6: Generate Gaussian matrix $\mathbf{G} \in \mathbb{R}^{d \times t}$ with entries drawn i.i.d. from $\mathcal{N}(0, 1/t^2)$, $t = \Theta(\log N)$, and Gaussian vector $g \sim \mathcal{N}(0, \mathbb{I}_d)$.
7: For all rows i of $\mathbf{J}_{\text{small}}$, set $\|(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g\|_2^2 = \alpha_i$, and
$$\tilde{\tau}_i = \begin{cases} 1 & \text{if } \alpha_i > 0 \\ \|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\Sigma^{-1}\mathbf{G}\|_2^2 & \text{otherwise} \end{cases}.$$
8: Using Algorithms 5 and 6, construct diagonal row sampling matrix $\mathbf{S} \in \mathbb{R}^{n_{\text{small}} \times n_{\text{small}}}$ such that $\mathbf{S}_{i,i} = \frac{1}{\sqrt{p_i}}$ with probability p_i, and $\mathbf{S}_{i,i} = 0$ otherwise, where $p_i \geq \min\left\{1, \frac{\log d}{\epsilon^2} \cdot \tilde{\tau}_i\right\}$
9: Return $\tilde{\mathbf{J}}$, where $\tilde{\mathbf{J}}$ is the result of stacking the matrices $\mathbf{S}'\tilde{\mathbf{J}}_{\text{big}}$ with $\mathbf{S}\mathbf{J}_{\text{small}}$.

The following lemma shows we can compute $\mathbf{S}\mathbf{J}_{\text{small}}$ in input sparsity time using Algorithm 5 and 6. We defer the proof of the Lemma to Section 7.1, and first show how our main results follow given Lemma 8.

Lemma 8. *Set the value $\gamma = 1$ in the dense case, and $\gamma = d$ in the sparse case. Let $\mathbf{J}_{\text{small}} \in \mathbb{R}^{n_{\text{small}} \times d}$ and let $\tilde{\mathbf{J}}_{\text{small}} \in \mathbb{R}^{m \times d}$ be the subset of rows of $\mathbf{J}_{\text{small}}$ constructed in Algorithm 4, where $m = \Theta((n_1 + n_2)/\gamma)$. Let $\mathbf{S}$ be the diagonal sampling matrix as constructed in Algorithm 4. Then with probability $1 - 1/d$, we have that $\mathbf{S}\mathbf{J}_{\text{small}}$ is an ϵ-subspace embedding for $\mathbf{J}_{\text{small}}$. Moreover, $\mathbf{S}$ has at most $\tilde{O}(d^2\gamma^2/\epsilon^2)$ non-zero entries, and $\mathbf{S}\mathbf{J}_{\text{small}}$ can be computed in time $\tilde{O}(\texttt{nnz}(\mathbf{T}_1) + \texttt{nnz}(\mathbf{T}_2) + d^3\gamma^2/\epsilon^2)$.*

We now prove our main theorem for in-database subspace embeddings, given Lemma 8.

Theorem 1 [In-Database Subspace Embedding] *Suppose $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{n \times d}$ is a join of two tables, where $\mathbf{T}_1 \in \mathbb{R}^{n_1 \times d_1}, \mathbf{T}_2 \in \mathbb{R}^{n_2 \times d_2}$. Then Algorithm 4 outputs a sketching matrix $\mathbf{S}^* \in \mathbb{R}^{k \times n}$ with $k = \tilde{O}(d^2\gamma^2/\epsilon^2)$ (where γ is chosen as in Lemma 8) such that $\tilde{\mathbf{J}} = \mathbf{S}^*\mathbf{J}$ is an ϵ-subspace embedding for $\mathbf{J}$, meaning*

$$\|\mathbf{S}^*\mathbf{J}x\|_2^2 = (1 \pm \epsilon)\|\mathbf{J}x\|_2^2$$

for all $x \in \mathbb{R}^d$ with probability at least $9/10$. The runtime to return $\mathbf{S}^*\mathbf{J}$ is the minimum of $\tilde{O}((n_1+n_2)d/\epsilon^2 + d^3/\epsilon^2)$ and $\tilde{O}((\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2))/\epsilon^2 + d^5/\epsilon^2)$.

Proof. Our algorithm partitions the rows of $\mathbf{J}$ into those from $\mathcal{B}_{\text{small}}$ and $\mathcal{B}_{\text{big}}$, and outputs the result of stacking sketches for $\mathbf{J}_{\text{small}}$ and $\mathbf{J}^{(i)}$ for each $i \in \mathcal{B}_{\text{big}}$. Thus it suffices to show that each sketch $\mathbf{S}^i\mathbf{J}^{(i)}$ is a subspace embedding for $\mathbf{J}^{(i)}$, and $\mathbf{S}\mathbf{J}_{\text{small}}$ is a subspace embedding for $\mathbf{J}_{\text{small}}$. The latter holds by Lemma 8, and the former follows directly from applying Lemma 5 and a union bound over the at most $O(n)$ such i. Since each such $\mathbf{J}^{(i)}$ can be written as $\mathbf{J}^i = \left(\hat{\mathbf{T}}_1^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_2^{(i)}\right) \in \mathbb{R}^{s_{(i)} \times d}$, the Fast Tensor-sketch lemma can be applied to $\mathbf{J}^{(i)}$ in time $\tilde{O}((\mathtt{nnz}(\mathbf{T}_1^{(i)}) + \mathtt{nnz}(\mathbf{T}_2^{(i)}) + d^2)/\epsilon^2)$. Note that $|\mathcal{B}_{\text{big}}| \leq 2(n_1+n_2)/(d\gamma)$, since there can be at most this many values of $s_{(i)}^1 + s_{(i)}^2 > d\gamma$. Thus the total running time is $\tilde{O}(\sum_{i \in \mathcal{B}_{\text{big}}}(\mathtt{nnz}(\mathbf{T}_1^{(i)}) + \mathtt{nnz}(\mathbf{T}_2^{(i)}) + d^2)/\epsilon^2) = \tilde{O}((\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2))/\epsilon^2 + (n_1+n_2)d/(\gamma\epsilon^2))$. Finally, applying CountSketch will cost $\tilde{O}((n_1+n_2)/(d\gamma) \cdot d/\epsilon^2 \cdot d)$, which is $\tilde{O}((n_1+n_2)d/\epsilon^2)$ for the dense case and $\tilde{O}((n_1+n_2)/\epsilon^2)$ for the sparse case. The remaining runtime analysis follows from Lemma 8, setting γ to be either 1 or d.

□

We now demonstrate how our subspace embeddings can be easily applied to obtain fast algorithms for regression. To do this, we will first need the following proposition, which shows that $w^T\mathbf{J}^T\mathbf{J}$ can be computed in input sparsity time for any $w \in \mathbb{R}^d$.

Proposition 4. *Suppose $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{n \times d}$ is a join of two tables, where $\mathbf{T}_1 \in \mathbb{R}^{n_1 \times d_1}, \mathbf{T}_2 \in \mathbb{R}^{n_2 \times d_2}$. Let $b \in \mathbb{R}^n$ be any column of the join $\mathbf{J}$. Let $U \subseteq [d]$ be any subset, and let $\mathbf{J}_U \in \mathbb{R}^{n \times |U|}$ be the subset of the columns of $\mathbf{J}$ contained in $\mathbf{U}$. Let $w \in \mathbb{R}^d$ be any vector, and let $x = \mathbf{J}w \in \mathbb{R}^n$. Then given w, the vector $x^T\mathbf{J}_U = w^T\mathbf{J}^T\mathbf{J}_U \in \mathbb{R}^{|U|}$ can be computed in time $O(\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2))$.*

Proof. Fix any $i \in \mathcal{B}(\mathbf{J})$, and similarly let $\mathbf{J}_U^{(i)}, \hat{\mathbf{T}}_{1,U}^{(i)}, \hat{\mathbf{T}}_{2,U}^{(i)}$ be $\mathbf{J}^{(i)}, \hat{\mathbf{T}}_1^{(i)}, \hat{\mathbf{T}}_2^{(i)}$ respectively, restricted to the columns of U. Note that we have $\mathbf{J}_U^{(i)} = (\hat{\mathbf{T}}_{1,U}^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),2}} \otimes \hat{\mathbf{T}}_{2,U}^{(i)}) \in \mathbb{R}^{s_i \times |U|}$. Let $x^{(i)} \in \mathbb{R}^{s_{(i)}}$ be x restricted to the rows inside of block $i \in \mathcal{B}(\mathbf{J})$. Since $x = \mathbf{J}w$ for some $w \in \mathbb{R}^d$, we have $x^{(i)} = \mathbf{J}^{(i)}w = (\hat{\mathbf{T}}_1^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_2^{(i)})w$. Then we have

$$\begin{aligned}
x^T\mathbf{J}_U &= \sum_{i \in \mathcal{B}} (x^{(i)})^T \mathbf{J}_U^{(i)} \\
&= \sum_{i \in \mathcal{B}} w^T \left(\hat{\mathbf{T}}_1^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_2^{(i)}\right)^T \\
&\cdot \left(\hat{\mathbf{T}}_{1,U}^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_{2,U}^{(i)}\right) \\
&= \sum_{i \in \mathcal{B}} w^T \Big(s_{(i),2}(\hat{\mathbf{T}}_1^{(i)})^T\hat{\mathbf{T}}_{1,U}^{(i)} + ((\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}}) \\
&\otimes ((\mathbf{1}^{s_{(i),2}})^T\hat{\mathbf{T}}_{2,U}^{(i)}) + (\mathbf{1}^{s_{(i),1}})^T\hat{\mathbf{T}}_{1,U}^{(i)} \otimes ((\hat{\mathbf{T}}_2^{(i)})^T\mathbf{1}^{s_{(i),2}}) \\
&+ s_{(i),1}(\hat{\mathbf{T}}_2^{(i)})^T\hat{\mathbf{T}}_{2,U}^{(i)} \Big)
\end{aligned} \tag{1}$$

where the last equality follows from the mixed product property of Kronecker products (see e.g., (Van Loan, 2000)). First note that the products $w^T s_{(i),2}(\hat{\mathbf{T}}_1^{(i)})^T\hat{\mathbf{T}}_{1,U}^{(i)}$ and $w^T s_{(i),1}(\hat{\mathbf{T}}_2^{(i)})^T\hat{\mathbf{T}}_{2,U}^{(i)}$ can be computed in $O(\mathtt{nnz}(\mathbf{T}_1^{(i)}) + \mathtt{nnz}(\mathbf{T}_2^{(i)}))$ time by computing the vector matrix product first. Thus, it suffices to show how to compute $w^T((\mathbf{1}^{s_{(i),1}})^T\hat{\mathbf{T}}_{1,U}^{(i)}) \otimes ((\hat{\mathbf{T}}_2^{(i)})^T\mathbf{1}^{s_{(i),2}})$ and $w^T((\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}}) \otimes ((\mathbf{1}^{s_{(i),2}})^T\hat{\mathbf{T}}_{2,U}^{(i)})$ quickly. By reshaping the Kronecker products (Van Loan, 2000), we have $w^T(\mathbf{1}^{s_{(i),1}})^T\hat{\mathbf{T}}_1^{(i)} \otimes ((\hat{\mathbf{T}}_{2,U}^{(i)})^T\mathbf{1}^{s_{(i),2}}) = ((\hat{\mathbf{T}}_{2,U}^{(i)})^T\mathbf{1}^{s_{(i),2}}w^T(\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}})^T$ Now $w^T(\hat{\mathbf{T}}_1^{(i)})^T$ can be computed in $O(\mathtt{nnz}(\mathbf{T}_1^{(i)}))$ time, at which point $w^T(\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}}$ can be computed in $O(s_{(i),1}) = O(\mathtt{nnz}(\mathbf{T}_1^{(i)}))$ time. Next, we can compute $\mathbf{1}^{s_{(i),2}}(w^T(\hat{\mathbf{T}}_1^{(i)})^T \mathbf{1}^{s_{(i),1}})$ in $O(s_{(i),2}) = O(\mathtt{nnz}(\mathbf{T}_2^{(i)}))$ time. Finally, $(\hat{\mathbf{T}}_{2,U}^{(i)})^T(\mathbf{1}^{s_{(i),2}}w^T \cdot (\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}})$ can be computed in $O(\mathtt{nnz}(\mathbf{T}_2^{(i)}))$ time. A similar argument holds for computing $w^T(\mathbf{1}^{s_{(i),1}})^T\hat{\mathbf{T}}_{1,U}^{(i)} \otimes ((\hat{\mathbf{T}}_2^{(i)})^T\mathbf{1}^{s_{(i),2}})$, which completes the proof, noting that $\sum_{i \in \mathcal{B}} \mathtt{nnz}(\mathbf{T}_1^{(i)}) + \mathtt{nnz}(\mathbf{T}_2^{(i)}) = \mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2)$.

□

We now state our main theorem for machine precision regression. We remark again that the success probability can be boosted to $1 - \delta$ by boosting the success probability of the corresponding subspace embedding to $1 - \delta$, as described earlier.

Theorem 2 [In-Database Regression (Theorem 2)] *Suppose $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{n \times d}$ is a join of two tables, where*

$T_1 \in \mathbb{R}^{n_1 \times d_1}, T_2 \in \mathbb{R}^{n_2 \times d_2}$. Let $\mathbf{U} \subset [d]$ be any subset, and let $\mathbf{J}_U \in \mathbb{R}^{N \times |U|}$ be $\mathbf{J}$ restricted to the columns in U, and let $b \in \mathbb{R}^n$ be any column of the join $\mathbf{J}$. Then there is an algorithm which returns $\hat{x} \in \mathbb{R}^{|U|}$ such that with probability $9/10$ we have

$$\|\mathbf{J}_U \hat{x} - b\|_2 \leq (1+\epsilon) \min_{x \in \mathbb{R}^{|U|}} \|\mathbf{J}_U x - b\|_2$$

The runtime required to compute $\hat{x}$ is the minimum of
$\tilde{O}\left(((n_1+n_2)d + d^3) + (\texttt{nnz}(\mathbf{T}_1) + \texttt{nnz}(\mathbf{T}_2) + d^2)\log(1/\epsilon)\right)$
and
$\tilde{O}\left(d^5 + (\texttt{nnz}(\mathbf{T}_1) + \texttt{nnz}(\mathbf{T}_2) + d^2)\log(1/\epsilon)\right)$.

Proof. The following argument follows a standard reduction from having a subspace embedding to obtaining high precision regression (see, e.g., Section 2.6 of (Woodruff et al., 2014)). We first compute a subspace embedding $\tilde{\mathbf{J}}_=\mathbf{S}^*\mathbf{J}$ for $\mathbf{J}$ via Theorem 1 with precision parameter $\epsilon_0 = \Theta(1)$, so that $\mathbf{S}^*\mathbf{J}$ has $k = \tilde{O}(d^2\gamma^2)$ rows. Note that in particular this implies that $\mathbf{S}^*\mathbf{J}_U$ is an ϵ_0-subspace embedding for $\mathbf{J}_U$. We then generate an OSNAP matrix $\mathbf{W} \in \mathbb{R}^{\tilde{\Theta}(d) \times k}$ via Lemma 6 with precision ϵ_0, and condition on the fact that $\mathbf{W}\mathbf{S}^*\mathbf{J}_U$ is an ϵ_0-subpsace embedding for $\mathbf{S}^*\mathbf{J}_U$, which holds with large constant probability, from which it follows that $\mathbf{W}\mathbf{S}^*\mathbf{J}_U$ is an $O(\epsilon_0)$-subspace embedding for $\mathbf{J}_U$. We then compute the QR factorization $\mathbf{W}\mathbf{S}^*\mathbf{J}_U = \mathbf{Q}\mathbf{R}^{-1}$, which can be done in $O(d^\omega)$ time via fast matrix multiplication (Demmel et al., 2007). By standard arguments (Woodruff et al., 2014), the matrix $\mathbf{J}_U\mathbf{R}$ is now $O(1)$-well conditioned – namely, we have $\sigma_{\max}(\mathbf{J}_U\mathbf{R})/\sigma_{\min}(\mathbf{J}_U\mathbf{R}) = O(1)$. Given this, we can apply the gradient descent update $x_{t+1} \leftarrow x_t + \mathbf{R}^T\mathbf{J}_U^T(b - \mathbf{J}_U\mathbf{R}x_t)$, which can be computed in $O(\texttt{nnz}(\mathbf{T}_1) + \texttt{nnz}(\mathbf{T}_2) + d^2)$ time via Proposition 4 (note we compute $\mathbf{R}x_t$ in $O(d^2)$ time first, and then compute $(\mathbf{J}_U\mathbf{R}x_t)$. Here we use the fact that $b - \mathbf{J}_U\mathbf{R}x_t = \mathbf{J}w$ for some w which is a vector in the column span of $\mathbf{J}$, and moreover, we can determine the value of w from computing $\mathbf{R}x_t$ and noting that $b = \mathbf{J}e_{j^*}$, where $j^* \in [d]$ is the index of b in $\mathbf{J}$. Since $\mathbf{R}^T\mathbf{J}_U^T$ is now well -conditioned, gradient descent now converges in $O(\log 1/\epsilon)$ iterations given that we have a constant factor approximation x_0 (Woodruff et al., 2014). Specifically, it suffices to have an $x_0 \in \mathbb{R}^d$ such that $\|\mathbf{J}_U x_0 - b\|_2 \leq (1+\epsilon_0)\min_{x \in \mathbb{R}^d} \|\mathbf{J}_U x - b\|_2$. But recall that such an x_0 can be obtained by simply solving $x_0 = \arg\min_x \|\mathbf{S}^*\mathbf{J}_U x - \mathbf{S}^* b\|_2$, using the fact that $\mathbf{S}^*$ is an ϵ_0- subspace embedding for the span of $\mathbf{J}$, which completes the proof of the theorem. □

7.1. Proof of Lemma 8

We start our proof by showing Algorithm 6 can sample one row with probability according to the ℓ_2 norm quickly after running the pre-processing Algorithm 5.

Algorithm 5 Pre-processing step for fast ℓ_2 sampling from rows of $\mathbf{J} \cdot \mathbf{Y}$, where $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ and $\mathbf{Y} \in \mathbb{R}^{d \times r}$.

1: For each block $i \in \mathcal{B}$, compute $a_{(i)} = \hat{T}_1^{(i)}\mathbf{Y}$ and $b_{(i)} = \hat{T}_2^{(i)}$
2: For each block $i \in \mathcal{B}$, construct a binary tree $\tau_{(i)}(\mathbf{T}_1), \tau_{(i)}(\mathbf{T}_2)$ as follows:
3: Each node $v \in \tau_{(i)}(\mathbf{T}_j)$ is a vector $v \in \mathbb{R}^{3r}$
4: If v is not a leaf, then $v = v_{\text{lchild}} + v_{\text{rchild}}$ with $v_{\text{lchild}}, v_{\text{rchild}}$ the left and right children of v.
5: The leaves of $\tau_{(i)}(\mathbf{T}_j)$ are given by the set

$$\{v_l^{(i),j} = \left(v_{l,1}^{(i),j}, v_{l,2}^{(i),j}, \ldots, v_{l,r}^{(i),j}\right) \in \mathbb{R}^{3r} \mid l \in [s_{(i),j}]\}$$

where $v_{l,q}^{(i),1} = (1, 2(a_{(i)})_{l,q}, (a_{(i)})_{l,q}^2) \in \mathbb{R}^3$ and $v_{l,q}^{(i),2} = ((b_{(i)})_{l,q}^2, (b_{(i)})_{l,q}, 1) \in \mathbb{R}^3$.
6: Compute the values $\langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle$ for all $i \in \mathcal{B}$, and also compute the sum $\sum_i \langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle$.

Algorithm 6 Sampling step for fast ℓ_2 sampling from rows of $\mathbf{J} \cdot \mathbf{Y}$.

1: Sample a block $i \in \mathcal{B}$ with probability

$$\frac{\langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle}{\sum_j \langle \text{root}(\tau_j(\mathbf{T}_1)), \text{root}(\tau_j(\mathbf{T}_2))\rangle}$$

2: Sample $l_1 \in [s_{(i),1}]$ with probability

$$p_{l_1} = \frac{\left\langle v_{l_1}^{(i),1}, \text{root}(\tau_i(\mathbf{T}_2))\right\rangle}{\sum_l \left\langle v_l^{(i),1}, \text{root}(\tau_j(\mathbf{T}_2))\right\rangle}$$

3: Sample $l_2 \in [s_{(i),2}]$ with probability

$$p_{l_2 \mid l_1} = \frac{\left\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\right\rangle}{\sum_l \left\langle v_{l_1}^{(i),1}, v_l^{(i),2}\right\rangle}$$

4: Return the row corresponding to (l_1, l_2) in block i.

Lemma 9. *Let $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{N\times d}$ be any arbitrary join on two tables, with $\mathbf{T}_i \in \mathbb{R}^{n_i\times d_i}$, and fix any $\mathbf{Y} \in \mathbb{R}^{d\times t}$. Then Algorithms 5 and 6, after an $O((\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2))(t+\log N))$-time pre-processing step (Algorithm 5), can produce samples $i^* \sim \mathcal{D}_{\mathbf{Y}}$ (Algorithm 6) from the distribution $\mathcal{D}_{\mathbf{Y}}$ over $[N]$ given by*

$$Pr_{i^*\sim\mathcal{D}_{\mathbf{Y}}}[i^* = j] = \frac{\|(\mathbf{J}\cdot\mathbf{Y})_{j,*}\|_2^2}{\|\mathbf{J}\cdot\mathbf{Y}\|_F^2}$$

such that each sample is produced in $O(\log N)$ time.

Proof. We begin by arguing the correctness of Algorithms 5 and 6. Let j^* be any row of $\mathbf{J}\cdot\mathbf{Y}$. Note that the row j^* corresponds to a unique block $i \in \mathcal{B} = \mathcal{B}(\mathbf{J})$, and two rows $l_1 \in [s_{(i),1}], l_2 \in [s_{(i),2}]$, such that $\mathbf{J}_{j^*,*} = (\hat{T}_1)_{l_1',*} + (\hat{T}_2)_{l_2',*}$, where $l_1' \in [n_1], l_2' \in [n_2]$ are the indices which correspond to l_1, l_2. For $l \in [s_{(i),j}]$, let $v_l^j, v_{l,q}^j$ be defined as in Algorithm 5. We first observe that if j^* corresponds to the block $i \in \mathcal{B}$, then $\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\rangle = \|(\mathbf{JY})_{j^*,*}\|_2^2$, since

$$\begin{aligned}
&\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\rangle \\
&= \sum_{q=1}^{r} v_{l_1,q}^{(i),1}\cdot v_{l_2,q}^{(i),2} \\
&= \sum_{q=1}^{r} (a_{(i)})_{l_1,q}^2 + 2(a_{(i)})_{l_1,q}(b_{(i)})_{l_2,q} + (b_{(i)})_{l_2,q}^2 \qquad (2)\\
&= \sum_{q=1}^{r} \left((a_{(i)})_{l_1,q} + (b_{(i)})_{l_2,q}\right)^2 \\
&= \|(\mathbf{JY})_{j^*,*}\|_2^2
\end{aligned}$$

where for each block $i \in \mathcal{B}$, we compute $a_{(i)} = \hat{T}_1^{(i)}\mathbf{Y}$ and $b_{(i)} = \hat{T}_2^{(i)}$ as defined in Algorithm 5. Thus it suffices to sample a row j^*, indexed by the tuple (i, l_1, l_2) where $i \in \mathcal{B}, l_1 \in [s_{(i),1}], l_2 \in [s_{(i),2}]$, such that the probability we sample j^* is given by $p_{j^*} = \langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\rangle / (\sum_{i',l_1',l_2'}\langle v_{l_1'}^{(i'),1}, v_{l_2'}^{(i'),2}\rangle)$. We argue that Algorithm 6 does precisely this. First note that for any $i \in \mathcal{B}$, we have

$$\langle v_{l_1}^{(i),1}, \mathrm{root}(\tau_i(\mathbf{T}_2))\rangle = \sum_{l_1\in[s_{(i),1}], l_2\in[s_{(i),2}]} \langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\rangle$$

Thus we first partition the set of all rows j^* by sampling a block i with probability $\frac{\langle \mathrm{root}(\tau_i(\mathbf{T}_1)), \mathrm{root}(\tau_i(\mathbf{T}_2))\rangle}{\sum_j \langle \mathrm{root}(\tau_j(\mathbf{T}_1)), \mathrm{root}(\tau_j(\mathbf{T}_2))\rangle}$, which is exactly the distribution over blocks induced by the ℓ_2 mass of the blocks. Conditioned on sampling $i \in \mathcal{B}$, it suffices now to sample l_1, l_2 from that block. To do this, we first sample l_1 with probability $\langle v_{l_1}^{(i),1}, \mathrm{root}(\tau_i(\mathbf{T}_2))\rangle / (\sum_l \langle v_l^{(i),1}, \mathrm{root}(\tau_j(\mathbf{T}_2))\rangle)$, which is precisely the distribution over indices $l_1 \in [s_{(i),1}]$ induced by the contribution of l_1 to the total ℓ_2 mass of block i. Similarly, once conditioned on l_1, we sample l_2 with probability $\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\rangle / (\sum_l \langle v_{l_1}^{(i),1}, v_l^{(i),2}\rangle)$, which is the distribution over indices $l_2 \in [s_{(i),2}]$ induced by the contribution of the row (l_1, l_2), taken over all l_2 with l_1 fixed. Taken together, the resulting sample $j^* \cong (i, l_1, l_2)$ is drawn from precisely the desired distribution.

Finally, we bound the runtime of this procedure. First note that computing $a_{(i)} = \hat{T}_1^{(i)}\mathbf{Y}$ and $b_{(i)} = \hat{T}_2^{(i)}$ for all blocks i can be done in $O(t(\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2)))$ time, since each row of the tables $\mathbf{T}_1, \mathbf{T}_2$ is in exactly one of the blocks, and each row is multiplied by exactly t columns of $\mathbf{Y}$. Once the $a_{(i)}, b_{(i)}$ are computed, each tree $\tau_{(i)}(\mathbf{T}_j)$ can be computed bottom up in time $O(\log N)$, giving a total time of $O(\log N(\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2)))$ for all trees. Given this, the values $\langle \mathrm{root}(\tau_i(\mathbf{T}_1)), \mathrm{root}(\tau_i(\mathbf{T}_2))\rangle$ can be computed in less than the above runtime. Thus the total pre-processing time is bounded by $O((t+\log N)(\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2)))$ as needed. For the sampling time, it then suffices to show that we can carry out Lines 2 and 3 in $O(\log N)$ time. But these samples can be samples from the root down, by first computing $\langle \mathrm{root}_{\mathrm{lchild}}(\tau_i(\mathbf{T}_1)), \mathrm{root}(\tau_i(\mathbf{T}_2))\rangle$ and $\langle \mathrm{root}_{\mathrm{rchild}}(\tau_i(\mathbf{T}_1)), \mathrm{root}(\tau_i(\mathbf{T}_2))\rangle$, sampling one of the left or right children with probability proportional to its size, and recursing into that subtree. Similarly, l_2 can be sampled by first computing $\langle v_{l_1}^{(i),1}, \mathrm{root}_{\mathrm{lchild}}(\tau_i(\mathbf{T}_2))\rangle$ and $\langle v_{l_1}^{(i),1}, \mathrm{root}_{\mathrm{rchild}}(\tau_i(\mathbf{T}_2))\rangle$ sampling one of the left or right children with probability proportional to its size, and recursing into that subtree. This completes the proof of the $O(\log N)$ runtime for sampling after pre-processing has been completed. ☐

Then we show how we can construct $\mathbf{S}$ by invoking Algorithm 5 and 6.

Lemma 10. *Let $\mathbf{J}_{small} \in \mathbb{R}^{n_{small}\times d}$ be the matrix constructed as in Algorithm 4 in the dense case. Then the diagonal sampling matrix $\mathbf{S}$, as defined within lines 4 through 8 of Algorithm 4, can be constructed in time $\tilde{O}(\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2) + d^2\gamma^2/\epsilon^2)$.*

Proof. We first show how we can quickly construct the matrix $\tilde{\mathbf{J}}_{\mathrm{small}}$, which consists of $m = \Theta((n_1+n_2)/\gamma)$ uniform samples from the rows of $\mathbf{J}_{\mathrm{small}}$. First, to sample the rows uniformly, since we already know the size of each block $s_{(i)}$, we can first sample a block $i \in \mathcal{B}_{\mathrm{small}}$ with probability proportional to its size, which can be done in $O(\log(|\mathcal{B}_{\mathrm{small}}|)) = O(\log N)$ time after the $s_{(i)}$'s are computed. Next, we can sample a row uniformly from T_i^j for each $j \in [2]$, and output the join of the two chosen rows, the result of which is a truly uniform row from $\mathbf{J}_{\mathrm{small}}$. Since we need m samples, and each sample has d columns, the overall runtime is $\tilde{O}((n_1+n_2)d/\gamma)$ to construct $\tilde{\mathbf{J}}_{\mathrm{small}}$, which is $O(\mathtt{nnz}(\mathbf{T}_1) + \mathtt{nnz}(\mathbf{T}_2))$ in the sparse case.

Once we have $\tilde{\mathbf{J}}_{\text{small}}$, we compute in line 4 of Algorithm 4 the sketch $\mathbf{W}\cdot\tilde{\mathbf{J}}_{\text{small}}$, where $\mathbf{W}\in\mathbb{R}^{t\times d}$ is the OSNAP Transformation of Lemma 6 with $\epsilon = 1/100$, where $t = \tilde{O}(d)$, which we can compute in $\tilde{O}(md) = \tilde{O}((n_1+n_2)d/\gamma)$ time by Lemma 6. Given this sketch $\mathbf{W}\cdot\tilde{\mathbf{J}}_{\text{small}}\in\mathbb{R}^{t\times d}$, the SVD of the sketch can be computed in time $O(d^\omega)$ (Demmel et al., 2007), where $\omega < 2.373$ is the exponent of fast matrix multiplication. Since $\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}$ is a $1/100$ subspace embedding for $\tilde{\mathbf{J}}$ with probability $99/100$ by Lemma 6, by Proposition 2 we have $\tau^{\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{A}) = (1\pm 1/100)\tau^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{A})$ for any matrix $\mathbf{A}$. Next, we can compute $\mathbf{V}\Sigma\mathbf{G}$ in the same $O(d^\omega)$ runtime, where $\mathbf{G}\in\mathbb{R}^{d\times t}$ is a Gaussian matrix with $t = \Theta(\log N)$ and with entries drawn independently from $\mathcal{N}(0,1/t^2)$. By standard arguments for Johnson Lindenstrauss random projections (see, e.g., Lemma 4.5 of (Li et al., 2013)), we have that $\|(x^T\mathbf{G})\|_2^2 = (1\pm 1/100)\|x\|_2^2$ for any fixed vector $x\in\mathbb{R}^d$ with probability at least $1-n^{-c}$ for any constant $c\geq 1$ (depending on t).

We now claim that $\tilde{\tau}_i$ as defined in Algorithm 4 satisfies $C^{-1}\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) \leq \tilde{\tau}_i \leq C\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ for some fixed constant $C\geq 1$. As noted above, $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) = (1\pm 1/100)\tau^{\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{A})$, so it suffices to show $C^{-1}\tau_i^{\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) \leq \tilde{\tau}_i \leq C\tau_i^{\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$. To see this, first note that if $(\mathbf{J}_{\text{small}})_{i,*}$ is contained within the row span of $\tilde{\mathbf{J}}_{\text{small}}$, then

$$\begin{aligned}\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) &= \|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\Sigma^{-1}\|_2^2\\ &= (1\pm 1/100)\|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\Sigma^{-1}\mathbf{G}\|_2^2\\ &= (1\pm 1/100)\tilde{\tau}_i\end{aligned}\tag{3}$$

Thus it suffices to show that if $(\mathbf{J}_{\text{small}})_{i,*}$ has a component outside of the span of $\tilde{J}_{\text{small}}$, then we have $\|(\mathbf{J}_{\text{small}})_{i,*}(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T)g\|_2^2 > 0$. To see this, note that $\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)$ is the projection onto the orthogonal space to the span of $\tilde{\mathbf{J}}_{\text{small}}$. Thus $\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g$ is a random non-zero vector in the orthogonal space of $\tilde{\mathbf{J}}_{\text{small}}$, thus $(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g \neq 0$ almost surely if $(\mathbf{J}_{\text{small}})_{i,*}$ has a component outside of the span of $\tilde{\mathbf{J}}_{\text{small}}$, which completes the proof of the claim.

Finally, and most significantly, we show how to implement line 8 of Algorithm 4, which carries out the the construction of $\mathbf{S}$. Given that $1/C\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) \leq \tilde{\tau}_i \leq C\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$, we can apply Theorem 1 of (Cohen et al., 2015), using that $n_{\text{small}} \leq (n_1+n_2)d\cdot\gamma$ via Proposition 3, which yields that $\sum_i\tilde{\tau}_i = O(n_{\text{small}}\frac{d}{m}) = O(d^2\gamma^2)$. Thus to construct the sampling matrix $\mathbf{S}$, it suffices to sample $\alpha = O(d^2\gamma^2\log d/\epsilon^2)$ samples from the distribution over the rows i of $\mathbf{J}_{\text{small}}$ given by $q_i = \frac{\tilde{\tau}_i}{\sum_i\tilde{\tau}_i}$. We now describe how to accomplish this.

We first show how to sample the rows i with $(\mathbf{J}_{\text{small}})_{i,*}$ $(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T)g = 0$. To do this, it suffices to sample rows from the distribution induced by the ℓ_2 norm of the rows of $(\mathbf{J}_{\text{small}})\mathbf{V}\Sigma^{-1}\mathbf{G}$. To do this, we can simply apply Algorithms 5 and 6 to the product $\mathbf{J}_{\text{small}}\cdot(\mathbf{V}\Sigma^{-1}\mathbf{G})$. First note that we can do this because $\mathbf{J}_{\text{small}}$ itself is a join of $(\mathbf{T}_1)_{\text{small}}$ and $(\mathbf{T}_2)_{\text{small}}$, which are just $\mathbf{T}_1,\mathbf{T}_2$ with all the rows contained in blocks $i\in\mathcal{B}_{\text{big}}$ removed. Since $\mathbf{V}\Sigma^{-1}\mathbf{G}\in\mathbb{R}^{d\times t}$ for $t = \tilde{O}(1)$, by Lemma 9 after $\tilde{O}(\mathtt{nnz}(\mathbf{T}_1)+\mathtt{nnz}(\mathbf{T}_2))$ time, for any $s\geq 1$ we can sample s times independently from this induced ℓ_2 distribution over rows in time $\tilde{O}(s)$. Altogether, we obtain the required $\alpha = O(d^2\gamma^2\log d/\epsilon^2)$ samples from the distribution over the rows i of $\mathbf{J}_{\text{small}}$ given by $q_i = \frac{\tilde{\tau}_i}{\sum_i\tilde{\tau}_i}$ in the case that $(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g = 0$, with total runtime $\tilde{O}(\mathtt{nnz}(\mathbf{T}_1)+\mathtt{nnz}(\mathbf{T}_2)+d^2\gamma^2/\epsilon^2)$.

Finally, for the case that $(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g > 0$, we can apply the same Algorithms 5 and 6 to sample from the rows of $\mathbf{J}_{\text{small}}\cdot\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g$. First observe, using the fact that $\sum_i\tilde{\tau}_i \leq O(d^2\gamma^2)$ by Theorem 1 of (Cohen et al., 2015), it follows that there are at most $O(d^2\gamma^2)$ indices i such that $\mathbf{J}_{\text{small}}\cdot\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g > 0$. Thus, when applying Algorithm 6 after the pre-processing step is completed, instead of sampling independently from the distribution induced by the norms of the rows, we can deterministically find all rows with $\mathbf{J}_{\text{small}}\cdot\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g > 0$ in $\tilde{O}(d^2\gamma^2)$ time, by simply enumerating over all computation paths of Algorithm 6 that occur with non-zero probability. Since there are $O(d^2\gamma^2)$ such paths, and each one is carried out in $O(\log N)$ time by Lemma 9, the resulting runtime is the same as the case where $(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g = 0$.

Finally, we argue that we can compute exactly the probabilities p_j with which we sampled a row j of $\mathbf{J}_{\text{small}}$, for each i that was sampled, which will be needed to determine the scalings of the rows of $\mathbf{J}_{\text{small}}$ that are sampled. For all the rows sampled with $(\mathbf{J}_{\text{small}})_{j,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g > 0$, the corresponding value of p_j is by definition 1. Note that if a row was sampled in both of the above cases, then it should in fact have been sampled in the case that $(\mathbf{J}_{\text{small}})_{j,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g > 0$, so we set $p_j = 1$. For every row j sampled via Algorithm 6 when $(\mathbf{J}_{\text{small}})_{j,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g = 0$, such that j corresponds to the tuple (i,l_1,l_2) where $i\in\mathcal{B}_{\text{small}}$ and $l_1\in[s_{(i),1}], l_2\in[s_{(i),2}]$, we can compute the probability it was sampled exactly via

$$p_j = p_{(i,l_1,l_2)} = \frac{\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\rangle}{\sum_{c\in\mathcal{B}}\langle \text{root}(\tau_c(\mathbf{T}_1)), \text{root}(\tau_c(\mathbf{T}_2))\rangle}$$

using the notation in Algorithm 6. Then setting $\mathbf{S}_{j,j} = \frac{1}{\sqrt{p_j}}$ for each sampled row j between both processes yields the desired construction of $\mathbf{S}$. □

With the above lemma in hand, we can give the proof of Lemma 8.

Proof of lemma 8. As argued in the proof of Lemma 10, we have that $(1/C)\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) \leq \tilde{\tau}_i \leq C\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ for some fixed constant $C \geq 1$ and all $i \in \mathcal{B}_{\text{small}}$. The result then follows immediately from Theorem 4 of (Cohen et al., 2015), using the fact that $n_{\text{small}}/m = \tilde{O}(d\gamma^2)$, where m is the number of rows subsampled in $\tilde{\mathbf{J}}_{\text{small}}$ from $\mathbf{J}_{\text{small}}$. □

8. General Join Queries

Below we introduce DB-Sketch as a class of algorithms, and show how any oblivious sketching algorithm that has the properties of DB-Sketch can be implemented efficiently for data coming from a join query. Since the required properties are very similar to the properties of linear sketches for Kronecker products, we will be able to implement them inside of a database. Given that the statistical dimension can be much smaller than the actual dimensions of the input data, our time complexity for ridge regression can be significantly smaller than that for ordinary least squares regression, which is important in the context of joins of many tables.

In the following we assume the join query is acyclic; nevertheless, for cyclic queries it is possible to obtain the hypergraph tree decomposition of the join and create a table for each vertex in the tree decomposition by joining the input tables that are a subset of the vertex's bag in the hypergraph tree decomposition. One can then replace the cyclic join query with an acyclic query using the new tables.

In our algorithm, we use FAQ and inside-out algorithms as subroutines. The definition of FAQ is given in Appendix 10. Let $\mathbf{J}' = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m$ be an acyclic join. Let ρ be a binary expression tree that shows in what order the algorithm inside-out (Abo Khamis et al., 2016) multiplies the factors for an arbitrary single-semiring FAQ; meaning, ρ has a leaf for each factor (each table) and $(m-1)$ internal nodes such that if two nodes have the same parent then their values are multiplied together during the execution of inside-out. We number the tables based on the order that a depth-first-search visits them in this expression tree. We let ρ denote the multiplication order of $\mathbf{J}'$.

Now that the ordering of the tables is fixed, we reformulate the Join table $\mathbf{J}'$ so that it can be expressed as a summation of tensor products. Assign each column c to one of the input tables that has c, and then let E_i denote the columns assigned to table $\mathbf{T}_i$, X_i be the projection of X onto E_i, and D_i be the domain of the tuples X_i (projection of $\mathbf{T}_i$ onto E_i).

Letting $N = |D_1||D_2|\ldots|D_m|$, we can reformulate $\mathbf{J}'$ as $\mathbf{J} \in \mathbb{R}^{N\times d}$ to have a row for any possible tuple $X \in D_1 \times \cdots \times D_m$. If a tuple X is present in the join, we put its value in the row corresponding to it, and if it is not present we put 0 in that row. Note that $\mathbf{J}$ has all the rows in $\mathbf{J}'$ and also may have many zero rows; however, we do not need to represent $\mathbf{J}$ explicitly, and the sparsity of $\mathbf{J}$ does not cause a problem. One key property of this formulation is that by knowing the values of a tuple x, the location of x in $\mathbf{J}$ is well-defined. Also note that since we have only added rows that are 0, any subspace embedding of $\mathbf{J}$ would be a subspace embedding of $\mathbf{J}'$, and for all vectors x, $\|\mathbf{J}x\|_2^2 = \|\mathbf{J}'x\|_2^2$.

Given a join query $\mathbf{J}$ with m tables and its multiplication order ρ, an oblivious sketching algorithm is an (m,ρ)-DB-Sketch if there exists a function $F : \mathbb{R}^{n\times d} \to R_F$ where R_F is the range of F and $F(\mathbf{A})$ represents the sketch of $\mathbf{A}$ in some form and has the following properties:

1. $F(\mathbf{A}_1 + \mathbf{A}_2) = F(\mathbf{A}_1) \oplus F(\mathbf{A}_2)$ where $\oplus$ is a commutative and associative operator.
2. For any V resulting from a Kronecker product of matrices $\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_m \in \mathbb{R}^{n_1\cdots n_m}$, $F(\mathbf{A}) = F_1(\mathbf{A}_1) \odot F_2(\mathbf{A}_2) \odot \cdots \odot F_m(\mathbf{A}_m)$ where $\odot$ is applied based on the ordering in ρ, and for all i, F_i has the same range as F, and $F_i(X_1 + X_2) = F_i(X_1) \oplus F_i(X_2)$. Furthermore, it should be possible to evaluate $F_i(v_i)$ in time $O(T_f \texttt{nnz}(v_i))$.
3. For any values A, B, C in the range of F, $A \odot (B \oplus C) = (A \odot B) \oplus (A \odot C)$

Theorem 11. *Given a join query $\mathbf{J}' = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m$ and a DB-Sketch algorithm, there exists an algorithm to evaluate $F(\mathbf{J}')$ in time $O(m(T_\odot + T_\oplus)T_{FAQ} + T_f mnd)$ where n is the size of the largest table and T_{FAQ} is the time complexity of running a single semiring FAQ, while $T_\odot$ and $T_\oplus$ are the time complexities of $\odot$ and $\otimes$, respectively.*

Corollary 12. *For any join query $\mathbf{J}$ with multiplication order ρ of depth $m-1$ and any scalar λ, let d_λ be the λ-statistical dimension of $\mathbf{J}$. Then there exists an algorithm that produces $\mathbf{S} \in \mathbb{R}^{k\times n}$ where $k = O(d_\lambda m^4/\epsilon^2)$ in time $O((mkd)T_{FAQ} + m^6 nd/\epsilon^2)$ such that with probability $1 - \frac{1}{\texttt{poly}(n)}$ simultaneously for all $x \in \mathbb{R}^d$*

$$\|\mathbf{SJ}x\|_2^2 + \lambda\|x\|_2^2 = (1 \pm \epsilon)(\|\mathbf{J}x\|_2^2 + \lambda\|x\|_2^2).$$

Proof. The proof follows by showing that the algorithm in Lemma 5 is a DB-Sketch. We demonstrate this by introducing the functions F_i and the operators $\oplus$ and $\odot$. The function $F_i(v_i)$ is an OSNAP transform (Lemma 6) of v_i, $\mathbf{A} \odot \mathbf{B}$ is $\mathbf{S}(\mathbf{A} \otimes \mathbf{B})$ where $\mathbf{S}$ is the Tensor Subsampled Randomized Hadamard Transform as defined in (Ahle et al., 2020), and $\mathbf{A} \oplus \mathbf{B}$ is a summation of tensors $\mathbf{A}$ and $\mathbf{B}$. Then it is easy to see that all the properties hold since Kronecker product distributes over summation.

Since F_i needs to be calculated for all rows in each table $\mathbf{T}_i$, which takes $O(T_f mnd)$ time, the operator $\oplus$ takes $O(kd)$ time to apply since the size of the sketch is $k \times d$. The operator $\otimes$ takes at most $O(kd)$ time to apply using the Fast

Fourier Transform (FFT) (Ahle et al., 2020). Therefore, the total time complexity can be bounded by $O((mkd)T_{\text{FAQ}} + m^6 nd/\epsilon^2)$.

Lastly, we remark that the algorithm in (Ahle et al., 2020) requires that the $\odot$ operator be applied to the input tensors in a binary fashion; however, it is shown in a separate version (Ahle & Knudsen, 2019) of the paper (Ahle et al., 2020) that the sketching construction and results of (Ahle et al., 2020) continue to hold when the tensor sketch is applied linearly. See Lemma 6 and 7 in (Ahle & Knudsen, 2019) and Lemma 10 in (Ahle et al., 2020). □

Corollary 12 gives an algorithm for all join queries when the corresponding hypertree decomposition is a path, or has a vertex for which all other vertices are connected to it. Although the time complexity for obtaining an ϵ-subspace embedding ($\lambda = 0$) is not better compared to the exact algorithm for ordinary least squares regression, for ridge regression it is possible to create sketches with many fewer rows and still obtain a reasonable approximation.

In the following we explain the algorithm for DB-Sketch using the FAQ formulation and inside-out algorithm (Abo Khamis et al., 2016). Let $\mathbf{J}_i$ denote the matrix resulting from keeping the column i of $\mathbf{J}$ and replacing all other columns with 0. Then we have $\mathbf{J} = \sum_i \mathbf{J}_i$. Using $\mathbf{J}_i$ we can define the proposed algorithm as finding $F(\mathbf{J}_i)$ for all tables $\mathbf{T}_i$ and then calculating $\oplus_i F(\mathbf{J}_i)$. Therefore, all we need to do is to calculate $F(\mathbf{J}_i)$ for all values of i. In the following we introduce an algorithm for the calculation of $F(\mathbf{J}_i)$ and then $F(\mathbf{J})$ is just the summation of the results for the different tables.

Let $e(X_k)$ be the D_k-dimensional unit vector that is 1 in the row corresponding to X_k, and let $v(X_k)$ be the $D_k \times d$ dimensional matrix that agrees with X_k in the row corresponding to X_k, and is 0 everywhere else.

Lemma 13. *For all tables,* $\mathbf{J}_i = \sum_{X \in \mathbf{J}'} e(X_1) \otimes \cdots \otimes e(X_{i-1}) \otimes v(X_i) \otimes e(X_{i+1}) \otimes \cdots \otimes e(X_m)$*, where* $\otimes$ *is the Kronecker product.*

Proof. For each tuple X, the term inside the summation has $N = |D_1||D_2|\ldots|D_m|$ rows and only 1 non-zero row because all of the tensors have only one non-zero row. The non-zero row is the row corresponding to X, and its value is the value of the same row in $\mathbf{J}_i$; therefore, $\mathbf{J}_i$ can be obtained by summing over all the tuples of $\mathbf{J}$. □

Lemma 14. *Let F be a DB-Sketch. Then $F(\mathbf{J}_i)$ can be computed in time $O((T_\odot + T_\oplus)T_{\mathit{FAQ}} + T_f ndm)$*

Proof. We define a FAQ for $F(\mathbf{J}_i)$ and then show how to calculate the result. Let $g_j(X_j) = F_j(e(X_j))$ for all $j \neq i$ and $g_i(X_i) = F_i(v(X_i))$. Note that the number of non-zero entries in $e(X_i)$ is at most d. Therefore, it is possible to find all values of $F_i(v(X_i))$ in time $O(T_f nd)$. The claim is it is possible to use the inside-out (Abo Khamis et al., 2016) algorithm for the following query and find $\mathbf{S}\mathbf{J}_i$:

$$\bigoplus_{X \in \mathbf{J}'} \bigodot_j g_j(X_j)$$

The mentioned query would be a FAQ if $\bigodot$ were commutative and associative. However, since we defined the ordering of the tables based on the multiplication order ρ, the inside-out algorithm multiplies the factors exactly in the same order needed for the DB-sketch algorithm. Therefore, we do not need the commutative and associative property of the $\odot$ operator to run inside-out.

Now we need to show that the query truly calculates $F(\mathbf{J}_i)$. Based on Lemma 13 and the properties of F we have:

$$\begin{aligned}
F(\mathbf{J}_i) =& F\Big(\sum_{X \in \mathbf{J}'} e(X_1) \otimes \ldots \\
& \otimes e(X_{i-1}) \otimes v(X_i) \otimes e(X_{i+1}) \otimes \ldots \\
& \otimes e(X_m)\Big) \\
=& \bigoplus_{X \in \mathbf{J}'} F(e(X_1) \otimes \ldots \\
& \otimes e(X_{i-1}) \otimes v(X_i) \otimes e(X_{i+1}) \otimes \ldots \\
& \otimes e(X_m)) \\
=& \bigoplus_{X \in \mathbf{J}'} \Big(F_1(e(X_1)) \odot \ldots \\
& \odot F_{i-1}(e(X_{i-1})) \odot F_i(v(X_i)) \odot F_{i+1}(e(X_{i+1})) \\
& \odot \cdots \odot F_m(e(X_m))\Big) \\
=& \bigoplus_{X \in \mathbf{J}'} \bigodot_j g_j(X_j)
\end{aligned}$$

as needed. □

Proof of Theorem 11. Finding the values of $F_i(X_i)$ for all i and all tuples of X_i takes $O(T_f ndm)$ time because F_i can be calculated in time $O(T_f \mathtt{nnz}(\mathbf{T}_i))$, and the total number of non-zero entries can be bounded by $O(ndm)$. The calculation of $F(\mathbf{J}_i)$ can be done in time $O(m(T_\odot + T_\oplus)T_{\text{FAQ}} + T_f ndm)$ using m rounds of the inside-out algorithm where $T_{FAQ} = O(md^2 n^{\text{fhtw}} \log(n))$ (Abo Khamis et al., 2016). After this step, we need to aggregate the results using the $\oplus$ operator to obtain the final result which takes $O(mT_\oplus)$ time. □

9. Connecting the Two Algorithms

In this paper, we have two algorithms for joins. One is for two-table joins and the other one is for general joins. Now

Table 3: General Algorithm on Two-Table Joins

	k	running time	relative error
$\mathbf{J}_1$	40	0.086	42.3%
	80	0.10	9.79%
	120	0.14	3.67%
	160	0.16	0.87%
	200	0.19	1.05%
$\mathbf{J}_2$	400	1.25	5.79%
	800	2.02	2.32%
	1200	2.93	1.85%
	1600	3.74	1.19%
	2000	4.50	0.96%

we will compare them in the context of two-table joins, both theoretically and empirically.

9.1. Theoretical Comparison

By Theorem 1, the total running time to obtain a subspace embedding is the minimum of $\tilde{O}((n_1+n_2)d/\epsilon^2+d^3/\epsilon^2)$ and $\tilde{O}((\mathtt{nnz}(\mathbf{T}_1)+\mathtt{nnz}(\mathbf{T}_2))/\epsilon^2+(n_1+n_2)/\epsilon^2+d^5/\epsilon^2)$ using Algorithm 4.

Now we consider the algorithm stated in Corollary 12. When running on the join of two tables, the algorithm is equivalent to applying the Fast Tensor-Sketch to each block and summing them up. Thus, the running time is $\tilde{O}(\sum_{i\in\mathcal{B}}(\mathtt{nnz}(\mathbf{T}_1^{(i)})/\epsilon^2+\mathtt{nnz}(\mathbf{T}_2^{(i)})/\epsilon^2+d^2/\epsilon^2)) = \tilde{O}((\mathtt{nnz}(\mathbf{T}_1)+\mathtt{nnz}(\mathbf{T}_2))/\epsilon^2+(n_1+n_2)d^2/\epsilon^2)$.

The running time of the algorithm in Corollary 12 is greater than the running time of Algorithm 4. In the extreme case the number of blocks can be really large, and even if each block has only a few rows we still need to pay an extra $\tilde{O}(d^2/\epsilon^2)$ time for it. This is the reason why we split the blocks into two sets ($\mathcal{B}_{\text{big}}$ and $\mathcal{B}_{\text{small}}$) and use a different approach when designing the algorithm for two-table joins.

9.2. Experimental Comparison

In our experiments we replace the Fast Tensor-Sketch with Tensor-Sketch. The theoretical analysis is similar since we still need to pay an extra $O(kd\log k)$ time to sketch a block for target dimension k.

We run the algorithm for the general case on joins $\mathbf{J}_1$ and $\mathbf{J}_2$ for different target dimensions. As shown in Table 3, in order to achieve the same relative error as Algorithm 4, we need to set a large target dimension and the running time would be significantly greater than it is in Table 1, even compared with the FAQ-based algorithm. This experimental result agrees with our theoretical analysis.

10. Database Background

We begin with some fundamental definitions relating to database joins.

Definition 15 (Join Hypergraph)**.** *Given a join $\mathbf{J}=\mathbf{T}_1 \bowtie \cdots \bowtie \mathbf{T}_m$, the hypergraph associated with the join is $H=(V,E)$ where V is the set of vertices and for every column c_i in J, there is a vertex v_i in V, and for every table $\mathbf{T}_i$ there is a hyper-edge e_i in E that has the vertices associated with the columns of $\mathbf{T}_i$.*

Definition 16 (Acyclic Join)**.** *We call a join query **acyclic** if one can repeatedly apply one of the two operations and convert the query to an empty query:*

1. *remove a column that is only in one table.*
2. *remove a table for which its columns are fully contained in another table.*

Definition 17 (Hypergraph Tree Decomposition)**.** *Let $H=(V,E)$ be a hypergraph and $T=(V',E')$ be a tree on a set of vertices, where each vertex $v'\in V'$ is called the **bag** of v', denoted by $b(v')$, and corresponds to a subset of vertices of V. Then T is called a **hypergraph tree decomposition** of H if the following holds:*

1. *for each hyperedge $e\in E$, there exists $v'\in V'$ such that $e\subseteq b(v')$, and*
2. *for each vertex $v\in V$, the set of vertices in V' that have v in their bag is non-empty and they form a connected subtree of T.*

Definition 18. *Let $H=(V,E)$ be a join hypergraph and $T=(V',E')$ be its tree decomposition. For each $v'\in V'$, let $X^{v'}=(x_1^{v'},x_2^{v'},\ldots,x_m^{v'})$ be the optimal solution to the following linear program: $\mathtt{min}\sum_{j=1}^{t}x_j$, subject to $\sum_{j:v_i\in e_j}x_j\geq 1, \forall v_i\in b(v')$ where $0\leq x_j\leq 1$ for each $j\in[t]$. Then the **width of** v' is $\sum_i x_i^{v'}$, denoted by $w(v')$, and the **fractional width of** T is $\max_{v'\in V'}w(v')$.*

Definition 19 (fhtw)**.** *Given a join hypergraph $H=(V,E)$, the **fractional hypertree width of** H, denoted by fhtw, is the minimum fractional width of its hypergraph tree decomposition. Here the minimum is taken over all possible hypertree decompositions.*

Observation 20. *The fractional hypertree width of an acyclic join is 1, and each bag in its hypergraph tree decomposition is a subset of the columns in some input table.*

Definition 21 (FAQ)**.** *Let $J=T_1\bowtie\cdots\bowtie T_m$ be a join of m input tables. For each table T_i, let $F_i: T_i\to S$ be a function mapping the rows of T_i to a set S. For every row $X\in J$, let X_i be the projection of X onto the columns of T_i. Then the following is a SumProd Functional Aggregation*

Query (FAQ):

$$\bigoplus_{X \in J} \bigotimes_{i} F_i(X_i) \tag{4}$$

where $(S, \oplus, \otimes)$ is a commutative semiring.

Theorem 22 ((Abo Khamis et al., 2016)). *Inside-out is an algorithm which computes the result of a FAQ in time $O(Tmd^2 n^{\mathit{fhtw}} \log(n))$ where m is the number of tables, d is the number of columns in J, n is the maximum number of rows in any input table, T is the time to compute the operators $\oplus$ and $\otimes$ on a pair of operands, and $fhtw$ is the fractional hypertree width of the query.*

In (Abo Khamis et al., 2018), given a join $J = T_1 \bowtie \cdots \bowtie T_m$, it is shown that the entries of $J^T J$ can be expressed as a FAQ and computed using the inside-out algorithm.