Efficient Algorithms for Vector Similarities

by

Sandeep B. Silwal

B.S., Massachusetts Institute of Technology (2019)
S.M., Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by:	Sandeep B. Silwal Department of Electrical Engineering and Computer Science May 17, 2024
Certified by:	Piotr Indyk Professor of Electrical Engineering and Computer Science, Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students

Efficient Algorithms for Vector Similarities

by

Sandeep B. Silwal

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

A key cog in machine learning is the humble embedding: vector representations of real world objects such as text, images, graphs, or molecules whose geometric similarities capture intuitive notions of semantic similarities. It is thus common to curate massive datasets of embeddings by inferencing on a machine learning model of choice. However, the sheer dataset size and large dimensionality is often *the* bottleneck in effectively leveraging and learning from this rich dataset. Inspired by this computational bottleneck in modern machine learning pipelines, we study the following question:

How can we efficiently compute on large scale high dimensional data?

In this thesis, we focus on two aspects of this question.

1. Efficient *local* similarity computation: we give faster algorithms for individual similarity computations, such as calculating notions of similarity between collections of vectors, as well as dimensionality reduction techniques which preserve similarities. In addition to computational efficiency, other resource constraints such as space and privacy are also considered.

2. Efficient *global* similarity analysis: we study algorithms for analyzing global relationships between vectors encoded in similarity matrices. Our algorithms compute on similarity matrices, such as distance or kernel matrices, without ever initializing them, thus avoiding an infeasible quadratic time bottleneck.

Overall, the main message of this thesis is that sublinear algorithms design principles are instrumental in designing scalable algorithms for big data.

Thesis supervisor: Piotr Indyk
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

Thank you Piotr for being a kind advisor and an infinite source of wisdom, inspiration, food, and jokes. I am incredibly lucky to have been your student.

Thank you Ronitt and Huy for serving on my committee and mentorship over the years. It was a pleasure working with you.

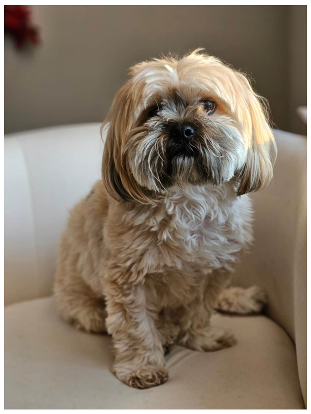
My fellow P.I.G.S. (Ainesh, Anders, Haike, Justin, Nicholas, Shyam, Tal, Talya, Vaggos, Ziv), thank you for the talks, laughs, problems, food, distractions, and friendships.

My dear friends Rikhav and Sushruth, we took many L's but we survived! Dear Hill Haus, thank you for your friendship over the years.

To MIT theory folks: thank you for such a welcoming environment, puzzles on hikes, and banter at the water machine. Many of us developed a coffee addiction together and I wouldn't have it any other way.

To my co-authors in graduate school, thank you for giving the the opportunity to work with such gifted researchers and lending me a bit of your vast knowledge. Working with you has been the highlight of my PhD journey.

Finally, my dear wife Darlene, this is your thesis as much as it is mine. You have given me strength in every step of the way, and I thank you from the bottom of my heart for your patience, love, understanding, support, and wit. This thesis is dedicated to you.



Thank you Maxwell (Maxi) Harsono.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last decade, the capabilities of machine learning (ML) models to meaningfully represent diverse data such as video, images, text, and molecules has dramatically increased. *Geometric* similarities between high dimensional embeddings from ML models are increasingly expressive enough to capture *semantic* relationships between real world objects. In response, practitioners frequently curate massive collections of vectors after running inference on neural networks, leading to large scale high dimensional datasets: in many industrial applications, it is common to have datasets of billions of embeddings in hundreds of dimensions. Due to the aforementioned property of embeddings, similarity computations lie at the heart of many ML tasks, such as search, retrieval, clustering, recommendations, and ranking.

The proliferation of such large datasets challenges our classical notions of algorithmic efficiency with regards to similarity computations. For example, quadratic time algorithms which compute on all pairs of vectors in a dataset are no longer feasible. As an illustration, consider running a quadratic sized computation on a dataset of ten billion vectors ($n = 10^{10}$). At the time of writing, a high end CPU can perform on the order of $\sim 10^{12}$ FLOP a second, meaning a n^2 computation requires time on the order of years. Ignoring computation time, even storing a quadratic sized answer requires exabytes of space. It stands to reason that efficient algorithms to process massive datasets are necessary. This motivates asking:

How can we efficiently compute on large scale high dimensional data?

The goal of this thesis is to respond to this formidable computational challenge.

We focus on two sides of the same coin: in the first part of the thesis, we study efficient algorithms for computing various notions of similarity between data points. Since our goal is to compute a single real number, we dub this part **local** similarity computation. In the second part, we focus on **global** similarity analysis and study algorithms for analyzing matrices which encode pairwise similarities between all vectors in a large dataset. Both parts are inspired by algorithmic design principles from *sublinear algorithms*, and many of the upper bounds are complemented with lower bounds, demonstrating that they lie precariously on the edge of algorithmic possibilities. Furthermore, all of our algorithms bridge the theory-practice divide and show significant empirical gains over prior methods.

1.1 Overview of Our Main Contributions

We briefly outline our main contributions. Our results are described in more detail in the proceeding sections. n is always the dataset size and d is the dimensionality.

1.1.1 Overview of Part 1: Efficient 'Local' Similarity Computation

Chamfer Distance: A Relaxation of Optimal Transport. In Chapter 3, we begin by considering notions of similarities that are defined on groups of vectors. We give an algorithm to compute the Chamfer distance, a common relaxation of the well studied Optimal Transport (OT) distance. Given two datasets A and B of size at most n, the Chamfer distance is the total cost to move every point in A to its nearest point in B. In comparison to OT, the mapping between A and B does not have to be one-to-one. In practice, the Chamfer distance is a popular proxy for the more computationally demanding OT, as the Chamfer distance admits a naive $O(n^2)$ time algorithm. However, this naive algorithm also cannot scale to large n. We resolve this by providing a *linear time* algorithm for computing the Chamfer distance. Our algorithm only computes an approximately optimal value and does not output the underlying mapping between A and B. This is not a limitation of our algorithm, but rather a fundamental obstacle, which we show using a (conditional) lower bound.

Dimensionality Reduction for Non-negative Sparse Vectors. The previous chapter focuses on speeding up the 'n' part of the computation. In Chapter 4, we shift focus to the 'd' parameter and study dimensionality reduction. Here, Euclidean dimensionality reduction (i.e. ℓ_2 distance) is a success story due to the Johnson-Lindenstrauss (JL) lemma which provides an efficient and practical map for embedding ℓ_2 in low-dimensions. Unfortunately, dimensionality reduction results are few and far between for other metrics, even other ℓ_p norms. We study an important setting where such results are algorithmically possible: the case of sparse vectors. Assuming our vectors are sparse, prior works give (linear) mappings which preserve ℓ_p distances between sparse vectors while projecting to a low dimension only depending on the sparsity and p. Unfortunately, such mappings necessarily embed to dimensions with exponential dependence on p.

In the chapter, we show that under the condition that our vectors are sparse *and non-negative*, we can embed our dataset to a dimension that only depends on the sparsity with no dependence on p. This allows us to obtain dimensionality reduction for large p such as the ℓ_∞ distance. Curiously, our dimensionality reduction scheme is non-linear and only works under the non-negativity assumption. Our (unconditional) lower bounds demonstrate that both assumptions (non-linear and non-negativity) are necessary.

Private Similarity Computation. Lastly, we focus on other notions of efficiency beyond computational time and study computing similarities with privacy constraints, formalized via the framework of differential privacy (DP). Many ML methods in privacy, such as DP model training, rely on computing the similarity between a query point (such as public or synthetic data) and private data. We abstract out this common subroutine and study the following fundamental algorithmic problem: Given a similarity function f and a large high-dimensional private dataset $X \subset \mathbb{R}^d$, output a differentially private data structure which approximates $\sum_{x \in X} f(x, y)$ for any query y. We consider the cases where f is a kernel function, such as $f(x, y) = e^{-\|x-y\|_2^2}$ (also known as DP kernel density estimation), or a distance function such as $f(x, y) = \|x - y\|_2$, among others.

Our theoretical results improve upon prior work and give better privacy-utility trade-offs, as well as faster query times for a wide range of kernels and distance functions. The unifying approach behind our results is leveraging 'low-dimensional structures' present in the specific functions f that we study, using tools such as provable dimensionality reduction, approximation theory, and one-dimensional decomposition of the functions.

The first part of the thesis is based on the following papers.

- [31]: *Near-Linear Time Algorithm for the Chamfer Distance*, joint with Ainesh Bakshi, Piotr Indyk, Rajesh Jayaram, and Erik Waingarten. Appeared in Advances in Neural Information Processing Systems 36 (NeurIPS 2023).

- *Improved dimensionality reduction for (non-negative) sparse vectors*, joint with David Woodruff and Richard Zhang.

- [25]: *Efficiently Computing Similarities to Private Datasets*, joint with Arturs Backurs, Zinan Lin, Sepideh Mahabadi, and Jakub Tarnawski. Appeared in the Twelfth International Conference on Learning Representations (ICLR 2024).

1.1.2 Overview of Part 2: Efficient 'Global' Similarity Analysis

In the second part, we shift our focus to global similarity analysis. Given a similarity function $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, we study algorithms for similarity matrices which encode all pairwise values $f(x, y)$ for all pairs $x, y \in \mathbb{R}^d$ in our dataset. Our algorithms never initialize these matrices, avoiding a quadratic time bottleneck.

Distance Matrices. In Chapter 6, we study the case where f is a distance function. Our results include efficient algorithms for computing matrix-vector products for a wide class of distance matrices, such as the ℓ_1 metric for which we get a linear runtime, as well as an $\Omega(n^2)$ lower bound for any algorithm which computes a matrix-vector product for the ℓ_∞ case, showing a separation between the ℓ_1 and the ℓ_∞ metrics. Our upper bound results, in conjunction with recent works on the matrix-vector query model, have many further downstream applications, including the fastest algorithm for computing a relative error low-rank approximation for the distance matrix induced by ℓ_1 and ℓ_2^2 functions and the fastest algorithm for computing an additive error low-rank approximation for the ℓ_2 metric, in addition to applications for fast matrix multiplication. We also give algorithms for constructing distance matrices and show that one can construct an approximate ℓ_2 distance matrix in time faster than the bound implied by the Johnson-Lindenstrauss lemma.

Kernel Matrices. Next we shift our focus to the case where f is a *kernel* function, for example $f(x, y) = \exp(-\|x-y\|_2)$. The main drawback of using kernel methods (learning and inference using kernel matrices) is again efficiency – given n input points, most kernel-based algorithms need to materialize the full $n \times n$ kernel matrix before performing any subsequent computation, thus incurring $\Omega(n^2)$ runtime. Breaking this quadratic barrier for various problems has therefore been a subject of extensive research efforts. We break the quadratic barrier and obtain *subquadratic* time algorithms for several fundamental linear-algebraic and graph processing primitives, including approximating the top eigenvalue and eigenvector and low-rank approximation. We build on the recently developed Kernel Density Estimation framework, which (after preprocessing in time subquadratic in n) can return estimates of row/column sums of the kernel matrix. In particular, we develop efficient reductions from *weighted vertex* and *weighted edge sampling* on kernel graphs, *simulating random walks* on kernel graphs, and *importance sampling* on matrices to Kernel Density Estimation and show that we can generate samples from these distributions in *sublinear* (in the support of the distribution) time. Our sampling primitives are the central ingredient in each of our applications.

The second part of the thesis is based on the following papers.

- [108]: *Faster Linear Algebra for Distance Matrices*, joint with Piotr Indyk. Appeared in Advances in Neural Information Processing Systems 35 (NeurIPS 2022).
- [30]: *Subquadratic Algorithms for Kernel Matrices via Kernel Density Estimation*, joint with Ainesh Bakshi, Piotr Indyk, Praneeth Kacham, and Samson Zhou. Appeared in the Eleventh International Conference on Learning Representations (ICLR 2023).

1.2 The Chamfer Distance: A Relaxation of OT

For any two point sets $A, B \subset \mathbb{R}^d$ of sizes up to n, the Chamfer distance[1] from A to B is defined as

$$\texttt{CH}(A, B) = \sum_{a \in A} \min_{b \in B} d_X(a, b)$$

where d_X is the underlying distance measure, such as the Euclidean (ℓ_2) or Manhattan (ℓ_1) distance. The Chamfer distance, and its weighted generalization called Relaxed Earth Mover Distance [17, 127], are popular measures of dissimilarity between point clouds. They are widely used in machine learning (e.g., [127, 193]), computer vision (e.g., [18, 80, 116, 184]) and computer graphics [137]. Subroutines for computing Chamfer distances are available in

[1]This is the definition adopted, e.g., in [18]. Some other papers, e.g., [80], replace each distance term $d_X(a, b)$ with its square, e.g., instead of $\|a - b\|_2$ they use $\|a - b\|_2^2$. In this paper we focus on the first definition, as it emphasizes the connection to Earth Mover Distance and its relaxed weighted version in [17, 127].

popular libraries, such as Tensorflow [3], Pytorch [2] and PDAL [1]. In many of those applications (e.g., [127]) Chamfer distance is used as a faster proxy for the more computationally demanding Earth-Mover (Optimal Transport) Distance.

Despite the popularity of Chamfer distance, the naïve algorithm for computing it has quadratic $O(n^2)$ running time, which makes it difficult to use for large datasets. Faster approximate algorithms can be obtained by performing n exact or approximate nearest neighbor queries, one for each point in A. By utilizing the state of the art approximate nearest neighbor algorithms, this leads to $(1+\varepsilon)$-approximate estimators with running times of $O\left(n(1/\varepsilon)^{O(d)} \log n\right)$ in low dimensions [15] or roughly $O(dn^{1+\frac{1}{2(1+\varepsilon)^2-1}})$ in high dimensions [13]. Alas, the first bound suffers from exponential dependence on the dimension, while the second bound is significantly subquadratic only for relatively large approximation factors.

We overcome this bottleneck and present the first $(1+\varepsilon)$-approximate algorithm for estimating Chamfer distance that has a *near-linear* running time, both in theory and in practice. Concretely, our contributions are as follows:

- When the underlying metric d_X is defined by the ℓ_1 or ℓ_2 norm, we give an algorithm that runs in time $O\left(nd\log(n)/\varepsilon^2\right)$ and estimates the Chamfer distance up to $1 \pm \varepsilon$ with 99% probability (see Theorem 3.1.1). In general, our algorithm works for any metric d_X supported by Locality-Sensitive Hash functions (see Definition 3.2.1), with the algorithm running time depending on the parameters of those functions. Importantly, the algorithm is quite easy to implement (see Algorithms 1 and 2 in Chapter 3).

- For the more general problem of *reporting* a mapping $g : A \to B$ whose cost $\sum_{a\in A} d_X(a, g(a))$ is within a factor of $1+\varepsilon$ from $\texttt{CH}(A, B)$, we show that, under a popular complexity-theoretic conjecture, an algorithm with a running time analogous to that of our *estimation* algorithm does not exist, even when $d_X(a, b) = \|a - b\|_1$. Specifically, under a Hitting Set Conjecture [198], any such algorithm must run in time $\Omega(n^{2-\delta})$ for any constant $\delta > 0$, even when the dimension $d = \Theta(\log^2 n)$ and $\varepsilon = \frac{\Theta(1)}{d}$. (In contrast, our estimation algorithm runs in near-linear time for such parameters). This demonstrates that, for the Chamfer distance, estimation is significantly easier than reporting.

- We experimentally evaluate our algorithm on real and synthetic data sets. Our experiments demonstrate the effectiveness of our algorithm for both low and high dimensional datasets and across different dataset scales. Overall, it is much faster (>**5x**) than brute force (even accelerated with KD-trees) and both faster and more sample efficient (**5-10x**) than simple uniform sampling. We demonstrate the scalability of our method by running it on *billion-scale* Big-ANN-Benchmarks datasets [176], where it runs up to **50x** faster than optimized brute force. In addition, our method is robust to different datasets: while uniform sampling performs reasonably well for some datasets in our experiments, it performs poorly on datasets where the distances from points in A to their neighbors in B vary significantly. In such cases, our algorithm is able to adapt its importance sampling probabilities appropriately and obtain significant improvements over uniform sampling.

1.3 Dimensionality Reduction for Sparse Vectors

Many popular algorithms for data processing in machine learning suffer from large running times on high-dimensional datasets. To alleviate this *curse of dimensionality*, a common paradigm is to first embed the data into a lower dimension and then run any desired algorithm in the embedded space. Arguably the most fundamental result in this area is the the Johnson-Lindenstrauss (JL) lemma. It states that any set of high-dimensional data X with $|X| = n$ can be embedded into an $O(\log n/\varepsilon^2)$-dimensional space while approximately preserving all pairwise ℓ_2 distances up to ε relative error [120]. While very versatile, the JL lemma is still a pessimistic worst case bound. This has led to a proliferation of works studying better trade-offs for specific problems with the goal of exploiting intrinsic structure within the problem or in the dataset to obtain smaller embedding dimensions. This 'fine-grained' approach has found success in many domains including nearest neighbor search [14, 107], clustering [33, 39, 114, 117, 146, 156], and numerical linear algebra [68, 201].

In this chapter, we consider *data sparsity* as the structure to exploit, rather than any specific algorithmic problem. Sparsity is an ubiquitous property of datasets and plays a crucial role in many tasks across machine learning, statistics, and signal-processing. Sparsity assumptions are additionally motivated by the fact that there exist dimensionality reduction *upper bounds* for sparse vectors for general ℓ_p norms. In contrast, virtually all the aforementioned progress beyond JL has been limited to the ℓ_2 norm[2].

The aim of this chapter is to shed further light on the landscape of dimensionality reduction for sparse vectors. We are interested in maps f which embed a set $X \subset \mathbb{R}^d$ of s-sparse vectors (vectors with at most s non-zero entries) and guarantee that all pairwise distances, measured in ℓ_p norm, between vectors in X are approximately preserved under f. The chapter explores the following natural questions. Building upon prior works which studied such maps for general ℓ_p norms, the first question asks:

> *(Q1) Can we obtain improved dimensional reduction for preserving pairwise distances between sparse vectors for general ℓ_p norms?*

Thus our first result of the chapter is an embedding for general sparse vectors.

Theorem 1.3.1. (Informal, see Theorem 4.4.2) *For every $p \geq 1$, there exists a linear map $f : \mathbb{R}^d \to \mathbb{R}^m$ for $m = s^{p+2}\log(s)\log(d/\varepsilon)2^{O(p)}/\varepsilon^2$ which satisfies $\|f(x)\|_p = (1 \pm \varepsilon)\|x\|_p$ for all s-sparse vectors $x \in \mathbb{R}^d$.*

In comparison, the best prior bound achieves embedding dimension $p^{O(p)}s^p\log^{p-1}(d)/\varepsilon^2$, also via a linear map [209]. While the exponent of s is smaller by 2 in the prior bound, it incurs an exponential dependence on p in both the terms $p^{O(p)}$ and $\log(d)^{p-1}$. Thus, qualitatively, we obtain improvements when the sparsity is small (for example $s \lesssim \log^{p-3}(d)$).

[2]This is not for the lack of trying: there exist fundamental limits disallowing for dimensionality reduction for general ℓ_p norms [43].

For general ℓ_p norms, it is known that any linear map must suffer an $\Omega(s^p)$ dependence on the dimension, which degrades as p increases [209]. In particular for the important ℓ_∞ case, all known bounds based on linear maps become vacuous. It is thus natural to explore if other conditions *in addition to* sparsity can overcome the exponential dependence on p. In our case, we look to sparse *non-negative* vectors, i.e., sparse vectors whose non-zero coordinates are positive. This additional constraint is motivated in two ways. Practically in many applications, many vector datasets are non-negative (for example embeddings produced by deep networks often get passed through a final ReLu layer). Theoretically, non-negativity has been exploited in other algorithms for sparse vectors, e.g. there exist separations between computing the convolution of non-negative sparse vectors and computing the convolution of general sparse vectors [41, 42, 118]. Thus we ask:

> *(Q2) Can we obtain improved dimensionality reduction for non-negative sparse vectors in general ℓ_p norms (such as ℓ_∞)?*

Note that one cannot assume non-negativity by simply shifting the dataset, as shifting can destroy sparsity. Thus, perhaps the most conceptually interesting contribution of the chapter is the following theorem for embedding non-negative sparse vectors.

Theorem 1.3.2. (Informal, see Theorem 4.5.2) *Let X be a set of n non-negative s-sparse vectors. For every $p \geq 1$, there exists a* ***non-linear*** *map $f : \mathbb{R}^d \to \mathbb{R}^m$ for $m = O(\log(n) \cdot \min(s^2/\varepsilon^2, s/\varepsilon^3))$ which satisfies $\|f(x) - f(y)\|_p = (1 \pm \varepsilon)\|x - y\|_p$ for all $x, y \in X$. For the ℓ_∞ case, we can instead guarantee $\|f(x) - f(y)\|_\infty = \|x - y\|_\infty$ for all $x, y \in X$ with $m = O(s \log(n))$.*

We now switch to lower bounds. As discussed in Chapter 4, our positive answer to Question (2) involves embedding with a *non-linear map*, and the embedding dimension has no dependence on p. This is most interesting in light of the fact that all prior dimensionality reduction upper bounds we are aware of (in virtually any context), use linear maps. The following question explores the role of linearity in embedding sparse vectors.

> *(Q3) What are the limitations of linear maps in dimensionality reduction for sparse vectors?*

Surprisingly, we show that our upper bound embedding of Theorem 4.5.2 is optimal in many ways; none of the following three hypothesis in the theorem statement can be dropped: (a) any such map f must be non-linear, (b) the input point set must be non-negative, and only the ℓ_p norms of *differences* can be preserved, not the sums. All mentioned lower bounds are proven in Chapter 4.

Additional results. The chapter includes additional consequences of our upper bounds to geometric optimization problems including clustering problems. We additionally study *average-case* embeddings, where a dimension reduction map is not required to approximate distances on all pairs of vectors, but rather only on a large constant fraction of pairs. The guarantees (upper and lower bounds) differ substantially compared to the setting where we require all pairs to be approximately preserved. See Chapter 4 for more details.

1.4 Privately Computing Similarities

It is evident that privacy is an important and often non-negotiable requirement in machine learning pipelines. Thus, privacy also serves as a resource which we must optimize for, in addition to computation time.

In response, the rigorous framework of differential privacy (DP) has been adopted as the de-facto standard for understanding and alleviating privacy concerns [76]. This is increasingly relevant as non-private ML models have been shown to profusely leak sensitive user information [47–51, 60, 84, 93, 189]. Many methodologies have been proposed in hopes of balancing DP requirements with retaining good downstream performance of ML models. Examples include generating public synthetic data *closely resembling* the private dataset at hand (and training on it) [139, 141, 142, 202, 204, 206], or selecting *similar* public examples for pre-training ML models [100, 205]. Furthermore, in the popular DP-SGD method, it is also widely understood that the use of public data bearing *similarity* to private datasets vastly improves downstream performance [72, 139, 202–204]. To list some concrete examples, [100] use a variant of the Fréchet distance to compute similarities of private and public data, [205] use a trained ML model to compute the similarity between public and private data, and [142] use a voting scheme based on the ℓ_2 distances between (embeddings of) private and synthetic data to select synthetic representatives.

Common among such works is the need to *compute similarities to a private dataset*. While this is explicit in examples such as [100, 142, 205], it is also implicit in many other works which employ the inductive bias that pre-training on *similar* public data leads to better DP model performance [72, 139, 202, 203].

We study the abstraction of this key subroutine and consider the following fundamental algorithmic problem: given a private dataset $X \subset \mathbb{R}^d$ and a similarity function $f(x, y) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, such as a kernel or distance function, output a *private* data structure $\mathcal{D}_X : \mathbb{R}^d \to \mathbb{R}$ which approximates the map $y \to \sum_{x \in X} f(x, y)$. We additionally require that $\mathcal{D}_X$ be always private with respect to X. Since $\mathcal{D}$ itself satisfies ε-DP, it can support an *arbitrary number* of queries without privacy loss. This is motivated by scenarios such as synthetic data generation, or when we do not have a pre-specified number of queries known upfront.

This problem has garnered much recent interest due to its strong motivation [7, 69, 94, 102, 192, 194]. It is a meaningful abstraction since similarities between (neural network based) embeddings can meaningfully represent rich relationships between objects such as images or text [167]. Indeed, beyond training private models, there is additional motivation from performing downstream tasks such as private classification or clustering, where a natural methodology is to classify a query as the class which it has the highest similarity to.

Discussion of Theoretical Results. The main trade-offs that we are interested in is between privacy, as measured with differential privacy (Definition 5.0.1), and accuracy of our answers, also called utility. For example, a data structure which always returns a fixed answer like 42 is clearly always private regardless of the number of queries performed, but is highly inaccurate. Thus, our goal is to obtain non-trivial accuracy guarantees while respecting privacy. Secondary, but important, concerns are query time and data structure construction time and space. Our main theoretical results are summarized in Table 5.1.

Theorem 1.4.1. (Informal; see Theorem 5.3.4 and Corollary 5.5.1) *Suppose the data points have bounded diameter in ℓ_1. For any $\alpha \in (0,1)$ and $\varepsilon > 0$, there exists an algorithm which outputs an ε-DP data structure $\mathcal{D}$ capable of answering any ℓ_1 distance query with α multiplicative error and $\tilde{O}\left(\frac{d^{1.5}}{\varepsilon\sqrt{\alpha}}\right)$ additive error in expectation. For the ℓ_2 case, where the points have bounded ℓ_2 diameter, we instead have $\tilde{O}\left(\frac{1}{\varepsilon\alpha^{1.5}}\right)$ additive error.*

Our approach is fundamentally different, and much simpler, than that of prior works [102], who used powerful black-box online learning results to approximate the sum of distances. Furthermore, given that we think of n as the largest parameter, we incur much smaller additive error. Our ℓ_1 upper bounds are complemented with a lower bound stating that any ε-DP algorithm supporting ℓ_1 distance queries for private datasets in the box $[0,R]^d$ must incur $\tilde{\Omega}(Rd/\varepsilon)$ error.

Theorem 1.4.2. (Informal; see Theorem 5.4.2) *Any ε-DP data structure which answers ℓ_1 distance queries with additive error at most T for any query must satisfy $T = \tilde{\Omega}(Rd/\varepsilon)$.*

We also obtain additional novel results for kernel functions. For example for the kernel $\frac{1}{1+\|x-y\|_2}$, we obtain the first private data structures; see Table 5.1 in the chapter for details.

Discussion of Empirical Results. Our experimental results are given in Section 5.10. The first experiment demonstrates that our ℓ_1 query algorithm is superior to prior state of the art [102] for accurately answering distance queries. The error of our algorithm smoothly decreases as ε increases, but their algorithms always return the trivial estimate of 0. This is due to the fact that the constants used in their theorem are too large to be practically useful.

We additionally explore an application to DP classification on the CIFAR-10 dataset. The standard setup is to train a private classification model on the training split (viewed as the private dataset), with the goal of accurately classifying the test split [72, 203]. Our methodology is simple, fast, and does not require a GPU: we simply instantiate a private similarity data structure for each class and assign any query to the class which it has the highest similarity to (or smallest distance if f is a distance). We set f to be ℓ_2^2 since it has arguably the simplest algorithm among the f's studied in the chapter. In contrast to prior works, our methodology involves no DP-SGD training. For comparable accuracy, we use > **3 orders of magnitude** less runtime compared to prior baselines [72, 203]. See Chapter 5 for further empirical evaluations.

1.5 Subquadratic Algorithms for Distance Matrices

Given a set of n points $X = \{x_1, \dots, x_n\}$, the distance matrix of X with respect to a distance function f is defined as the $n \times n$ matrix A satisfying $A_{i,j} = f(x_i, x_j)$. Distances matrices are ubiquitous objects arising in various applications ranging from learning image manifolds [188, 195], signal processing [179], biological analysis [99], and non-linear dimensionality reduction [70, 124, 125, 188], to name a few[3]. Unfortunately, explicitly computing and storing A requires at least $\Omega(n^2)$ time and space. Such complexities are prohibitive for scaling to large datasets.

A silver lining is that in many settings, the matrix A is not explicitly required. Indeed in many applications, it suffices to compute some underlying function or property of A, such as the eigenvalues and eigenvectors of A or a low-rank approximation of A. Thus an algorithm designer can hope to use the special geometric structure encoded by A to design faster algorithms tailored for such tasks.

Therefore, it is not surprising that many recent works explicitly take advantage of the underlying geometric structure of distance matrices, and other related families of matrices, to design fast algorithms (see Section 6.0.1 for a thorough discussion of prior works). In this work, we continue this line of research and take a broad view of algorithm design for distance matrices. Our main motivating question is the following:

> *Can we design algorithms for fundamental linear algebraic primitives which are specifically tailored for distance matrices and related families of matrices?*

We make progress towards the motivating question by studying three of the most fundamental primitives in algorithmic linear algebra. Specifically:

1. We study upper and lower bounds for computing matrix-vector products for a wide array of distance matrices:

   Creating efficient versions of matrix-vector queries for distance matrices automatically lends itself to many further downstream applications. We remark that our algorithms can access to the set of input points but *do not* explicitly create the distance matrix. A canonical example of our upper bound results is the construction of matrix-vector queries for the function $f(x, y) = \|x - y\|_p^p$.

   Theorem 1.5.1. *Let $p \geq 1$ be an integer. Suppose we are given a dataset of n points $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$. X implicitly defines the matrix $A_{i,j} = \|x_i - x_j\|_p^p$. Given a query $z \in \mathbb{R}^n$, we can compute Az exactly in time $O(ndp)$. If p is odd, we also require $O(nd \log n)$ preprocessing time.*

   We give similar guarantees for a wide array of functions f and we refer the reader to Table 6.1 which summarizes our matrix-vector query upper bound results.

   We also study fundamental limits for any upper bound algorithms. In particular, we show that *no algorithm* can compute a matrix-vector query for general inputs for the ℓ_∞ metric in subquadratic time, assuming a standard complexity-theory assumption called the *Strong Exponential Time Hypothesis (SETH)* [103, 104].

[3]We refer the reader to the survey [74] for a more thorough discussion of applications of distance matrices.

Theorem 1.5.2. *For any $\alpha > 0$ and $d = \omega(\log n)$, any algorithm for exactly computing Az for any input z, where A is the ℓ_∞ distance matrix, requires $\Omega(n^{2-\alpha})$ time (assuming SETH).*

This shows a separation between the functions listed in Table 6.1 and ℓ_∞.

2. We give fast algorithms for constructing distance matrices. To establish some context, recall that the classic JL lemma states (roughly) that a random projection of a dataset $X \subset \mathbb{R}^d$ of size n onto a dimension of size $O(\log n)$ approximately preserves all pairwise distances [119]. A common applications of this lemma is to *instantiate* the ℓ_2 distance matrix. A naive algorithm which computes the distance matrix after performing the JL projection requires approximately $O(n^2 \log n)$ time. Surprisingly, we show that the JL lemma is not tight with respect to creating an approximate ℓ_2 distance matrix; we show that one can initialize the ℓ_2 distance in an asymptotically better runtime.

   Theorem 1.5.3. (Informal; See Theorem 6.6.5) *We can calculate a $n \times n$ matrix B such that each (i, j) entry B_{ij} of B satisfies $(1-\varepsilon)\|x_i - x_j\|_2 \le B_{ij} \le (1+\varepsilon)\|x_i - x_j\|_2$ in time $O(\varepsilon^{-2} n^2 \log^2(\varepsilon^{-1} \log n))$.*

   Our result can be viewed as the natural runtime bound which would follow if the JL lemma implied an embedding dimension bound of $O(\text{poly}(\log\log n))$. While this is impossible, as it would imply an exponential improvement over the JL bound which is tight [131], we achieve our speedup by carefully reusing distance calculations via tools from metric compression [111]. Our results also extend to the ℓ_1 distance matrix (see Theorem 6.6.5).

3. Other results: Chapter 6 additionally contains results on *approximate* matrix multiplication for ℓ_∞, as well as downstream applications of our matrix vector product upper bounds including faster algorithms for eigenvector calculations and low-rank approximation. See Table 6.2 for details.

1.6 Subquadratic Algorithms for Kernel Matrices

For a kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and a set $X = \{x_1 \dots x_n\} \subset \mathbb{R}^d$ of n points, the entries of the $n \times n$ kernel matrix K are defined as $K_{i,j} = k(x_i, x_j)$. Alternatively, one can view X as the vertex set of a complete weighted graph where the weights between points are defined by the kernel matrix K. Popular choices of kernel functions k include the Gaussian kernel, the Laplace kernel, exponential kernel, etc; see [98, 173, 174] for a comprehensive overview.

Despite their wide applicability, kernel methods suffer from drawbacks, one of the main being efficiency – given n input points in d dimensions, many kernel-based algorithms need to materialize the full $n \times n$ kernel matrix K before performing the computation. For some problems this is unavoidable, especially if high-precision results are required [21]. In this work, we show that we can in fact break this $\Omega(n^2)$ barrier for several fundamental problems in numerical linear algebra and graph processing. We obtain algorithms that run in $o(n^2)$ time and scale inversely-proportional to the smallest entry of the kernel matrix. This allows us to skirt several known lower bounds, where the hard instances require the smallest kernel entry to be polynomially small in n. Our parameterization in terms of the smallest entry is motivated by the fact in practice, the smallest kernel value is often a fixed constant [23, 24, 122, 147, 177]. We build on recently developed fast approximate algorithms for Kernel Density Estimation [22, 23, 55, 58, 177]. Specifically, these papers present fast approximate data structures with the following functionality:

Definition 1.6.1. (Kernel Density Estimation (KDE) Queries) *For a given dataset $X \subset \mathbb{R}^d$ of size n, kernel function k, and precision parameter $\varepsilon > 0$, a KDE data structure supports the following operation: given a query $y \in \mathbb{R}^d$, return a value $\mathrm{KDE}_X(y)$ that lies in the interval $[(1-\varepsilon)z, (1+\varepsilon)\, z]$, where $z = \sum_{x \in X} k(x, y)$, assuming that $k(x, y) \geq \tau$ for all $x \in X$.*

The performance of the state of the art algorithms for KDE also scales proportional to the smallest kernel value of the dataset (see Table 2.1). In short, after a preprocessing time that is sub-quadratic (in n), KDE data structures use time sublinear in n to answer queries defined as above. Note that for all of our kernels, $k(x, y) \leq 1$ for all inputs x, y.

We show that given a KDE data structure as described above, it is possible to solve a variety of matrix problems in time subquadratic time $o(n^2)$, i.e., sublinear in the matrix size. We emphasize that in our applications, we only require *black-box* access to KDE queries. Given this, we design such algorithms for problems such as eigenvalue/eigenvector estimation, low-rank approximation, and graph sparsification.

Our results are obtained via the following two-pronged approach. First, we use KDE data structures to design algorithms for the following basic primitives, frequently used in sublinear time algorithms and property testing:

1. sampling vertices by their (weighted) degree in K (Theorems 7.3.2 and 7.3.4 and Algorithms 23 / 25),

2. sampling random neighbors of a given vertex by edge weights in K and sampling a random weighted edge (Theorem 7.3.5 and Algorithms 26 and 27),

3. performing random walks in the graph K (Theorem 7.3.7 and Algorithm 28), and

4. sampling the rows of the edge-vertex incident matrix and the kernel matrix K, both with probability proportional to respective row norms squared (Section 7.4.1, Theorem 7.4.1, and Section 7.4.2, Corollary 7.4.10 respectively).

In the second step, we use these primitives to implement a host of algorithms for the aforementioned problems. We emphasize that these primitives are used in a black-box manner, meaning that any further improvements to their running times will automatically translate into improved algorithms for the downstream problems. For our applications, we make the following parameterization, which we expand upon in Remark 7.2.1 and Section 7.2.1. At a high level, many of our applications, such as spectral sparsification, are succinctly characterized by the following parameterization.

Table 1.1: Summary of applications for KDE subroutines. We suppress dependence on the precision ε.

Problem	# of KDE Queries	Post-processing time	Prior Work
Spectral sparsification (Thm. 7.0.1)	$\widetilde{O}\left(\frac{n}{\tau^3}\right)$	$O\left(\frac{nd}{\tau^3}\right)$	Remark 7.2.1
Laplacian system solver (Thm. 7.0.1)	$\widetilde{O}\left(\frac{n}{\tau^3}\right)$	$O\left(\frac{nd}{\tau^3}\right)$	Remark 7.2.1
Low-rank approx. (Thm. 7.0.3)	$O(n)$	$O\left(n \cdot \text{poly}\left(k\right) + nkd\right)$	Remark 7.2.3
Approximating 1st Eigenvalue (Thm. 7.0.2)	Remark 7.2.2	$d \cdot \text{poly}(1/\tau)$	$\omega(n)$ (Remark 7.2.2)

Parameterization 1.6.1. *All of our algorithms are parameterized by the smallest edge weight in the kernel matrix, i.e., the smallest edge weight in the matrix K is at least τ.*

Table 1.1 lists the applications of our primitives, along with the number of KDE queries required in addition to any post-processing time. We refer to the specific sections in Chapter 7 for full details.

Empirical Results. We empirically demonstrate the efficacy of our algorithms on low-rank approximation (LRA) and spectral sparsification, where we observe a **9x** decrease in the number of kernel evaluations over baselines for LRA and a **41x** reduction in the graph size for spectral sparsification.

Chapter 2

Preliminaries

Throughout the thesis, any input dataset will be a set of n points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$. For $p \geq 1$, the ℓ_p norm of a vector $x \in \mathbb{R}^d$ is defined as $\|x\|_p = \left(\sum_{i=1}^d |x_i|^p\right)^{1/p}$ and $\|x\|_\infty = \max_i |x_i|$. Throughout the thesis, $\tilde{O}$ hides logarithmic factors in n and d.

Dimensionality Reduction Tools. The Johnson-Lindenstrauss (JL) lemma will make several appears throughout the thesis. It provides a dimensionality reduction for the ℓ_2 norm.

Theorem 2.0.1 ([119]). *Let $\varepsilon \in (0,1)$ and define $\boldsymbol{T} : \mathbb{R}^d \to \mathbb{R}^k$ by*

$$\boldsymbol{T}(x)_i = \frac{1}{\sqrt{k}} \sum_{j=1}^d Z_{ij} x_j, \quad i = 1, \ldots, k.$$

Then for every vector $x \in \mathbb{R}^d$, we have

$$\Pr[(1-\varepsilon)\|x\|_2 \leq \|\boldsymbol{T}(x)\|_2 \leq (1+\varepsilon)\|x\|_2] \geq 1 - e^{c\varepsilon^2 k},$$

Letting $k = O(\log(n)/\varepsilon^2)$ and using the union bound allows us to project a set of n datapoints into k dimensions while preserving all pairwise ℓ_2 distances up to a $1 \pm \varepsilon$ factor. The following modification of the JL map given above allows us to embed ℓ_2 into a low dimensional ℓ_1 space.

Theorem 2.0.2 ([148]). *Let $\varepsilon \in (0,1)$ and define $\boldsymbol{T} : \mathbb{R}^d \to \mathbb{R}^k$ by*

$$\boldsymbol{T}(x)_i = \frac{1}{\beta k} \sum_{j=1}^d Z_{ij} x_j, \quad i = 1, \ldots, k$$

where $\beta = \sqrt{2/\pi}$. Then for every vector $x \in \mathbb{R}^d$, we have

$$\Pr[(1-\varepsilon)\|x\|_2 \leq \|\boldsymbol{T}(x)\|_1 \leq (1+\varepsilon)\|x\|_2] \geq 1 - e^{c\varepsilon^2 k},$$

where $c > 0$ is a constant.

Table 2.1: Instantiations of KDE queries. The query times depend on the dimension d, (multiplicative) accuracy ε, and additive error τ (alternatively, assuming a lower bound of τ). The parameter β is assumed to be a constant and log factors are not shown.

Kernel	$f(x,y)$	Preprocessing Time	KDE Query Time	Reference
Gaussian	$e^{-\|x-y\|_2^2}$	$\frac{nd}{\varepsilon^2\tau^{0.173+o(1)}}$	$\frac{d}{\varepsilon^2\tau^{0.173+o(1)}}$	[58]
Exponential	$e^{-\|x-y\|_2}$	$\frac{nd}{\varepsilon^2\tau^{0.1+o(1)}}$	$\frac{d}{\varepsilon^2\tau^{0.1+o(1)}}$	[58]
Exponential	$e^{-\|x-y\|_2}$	$\frac{nd}{\varepsilon^2\tau^{0.5}}$	$\frac{d}{\varepsilon^2\tau^{0.5}}$	[23]
Laplacian	$e^{-\|x-y\|_1}$	$\frac{nd}{\varepsilon^2\tau^{0.5}}$	$\frac{d}{\varepsilon^2\tau^{0.5}}$	[23]
Rational Quadratic	$\frac{1}{(1+\|x-y\|_2^2)^\beta}$	$\frac{nd}{\varepsilon^2}$	$\frac{d}{\varepsilon^2}$	[22]

Kernel Density Estimation. For a kernel function k, such as those in Table 2.1, we define a kernel density estimation (KDE) query is defined as follows. Table 2.1 lists known bounds on datastructures which perform KDE queries for a wide range of kernels.

Definition 1.6.1. (Kernel Density Estimation (KDE) Queries) *For a given dataset $X \subset \mathbb{R}^d$ of size n, kernel function k, and precision parameter $\varepsilon > 0$, a KDE data structure supports the following operation: given a query $y \in \mathbb{R}^d$, return a value $\mathrm{KDE}_X(y)$ that lies in the interval $[(1-\varepsilon)z, (1+\varepsilon)\, z]$, where $z = \sum_{x\in X} k(x,y)$, assuming that $k(x,y) \geq \tau$ for all $x \in X$.*

Hardness Assumptions. Throughout the thesis, we rely on several fine-grained complexity hardness assumptions to prove lower bounds for computing similarities in various settings. The first conjecture is the following.

Definition 2.0.1. (Hitting Set (HS) problem) *The input to the problem consists of two sets of vectors $A, B \subseteq \{0,1\}^d$, and the goal is to determine whether there exists some $a \in A$ such that $a \cdot b \neq 0$ for every $b \in B$. If such an $a \in A$ exists, we say that a* hits *B.*

It is easy to see that the Hitting Set problem can be solved in time $O(n^2 d)$. The Hitting Set Conjecture [198] postulates that this running time is close to the optimal.

Conjecture 2.0.1. *Suppose $d = \Theta(\log^2 n)$. Then for every constant $\alpha > 0$, no randomized algorithm can solve the Hitting Set problem in $O(n^{2-\alpha})$ time.*

The second conjecture is the Strong Exponential Time Hypothesis (SETH) [103, 104]. It is a statement about the complexity of k-SAT, but we use a more convenient formulation in terms of the orthogonal vectors problem.

Definition 2.0.2. (Orthogonal Vectors problem (OVP)) *Given two sets of vectors $A = \{a^1, \dots, a^n\}$ and $B = \{b^1, \dots, b^n\}$, $A, B \subset \{0,1\}^d$, $|A| = |B| = n$, determine whether there exist $x \in A$ and $y \in B$ such that the dot product $x \cdot y = \sum_{j=1}^d x_j y_j$ (taken over reals) is equal to 0.*

Conjecture 2.0.2. [197] *Suppose $d = \omega(\log n)$. Assuming SETH, for every constant $\alpha > 0$, no randomized algorithm can solve OVP in $O(n^{2-\alpha})$ time.*

Part I

Efficient ‘Local’ Similarity Computation

Chapter 3

The Chamfer Distance

For any two point sets $A, B \subset \mathbb{R}^d$ of size up to n, recall that the Chamfer distance from A to B is defined as

$$\mathtt{CH}(A, B) = \sum_{a \in A} \min_{b \in B} d_X(a, b),$$

where d_X is the underlying distance measure (e.g., the Euclidean or Manhattan distance). The Chamfer distance is a popular measure of dissimilarity between point clouds, used in many machine learning, computer vision, and graphics applications, and admits a straightforward $O(dn^2)$-time brute force algorithm. Further, the Chamfer distance is often used as a proxy for the more computationally demanding Earth-Mover (Optimal Transport) Distance. However, the *quadratic* dependence on n in the running time makes the naive approach intractable for large datasets.

In this chapter, we overcome this bottleneck and present the first $(1 + \epsilon)$-approximate algorithm for estimating the Chamfer distance with a near-linear running time. Specifically, our algorithm runs in time $O(nd \log(n)/\epsilon^2)$ and is implementable. We also give evidence that if the goal is to *report* a $(1+\varepsilon)$-approximate mapping from A to B (as opposed to just its value), then any sub-quadratic time algorithm is unlikely to exist. Our experiments demonstrate that our upper bound algorithm is both accurate and fast on large high-dimensional datasets.

Our algorithm is given in Section 3.1, our lower bounds are proved in Section 3.3, and empirical evaluations are given in Section 3.4.

3.1 Algorithm and Road Map of Analysis

In this section, we establish our main result for estimating Chamfer distance:

Theorem 3.1.1 (Estimating Chamfer Distance in Nearly Linear Time)**.** *Given as input two datasets $A, B \subset \mathbb{R}^d$ such that $|A|, |B| \leq n$, and an accuracy parameter $0 < \varepsilon < 1$,* `Chamfer-Estimate` *runs in time $O(nd \log(n)/\varepsilon^2)$ and outputs an estimator η such that with probability at least* 99/100*,*

$$(1 - \varepsilon)\mathtt{CH}(A, B) \leq \eta \leq (1 + \varepsilon)\mathtt{CH}(A, B),$$

when the underlying metric is Euclidean (ℓ_2) or Manhattan (ℓ_1) distance.

Let us now motivate the algorithm design, with the formal algorithm presented in Algorithm 1. For ease of exposition, we make the simplifying assumption that the underlying metric is Manhattan distance, i.e. $d_X(a,b) = \|a-b\|_1$. Our algorithm still succeeds whenever the underlying metric admits a locality-sensitive hash function (see Definition 3.2.1).

Uniform vs Importance Sampling. A natural algorithm for estimating $\texttt{CH}(A,B)$ proceeds by *uniform sampling*: sample an $a \in A$ uniformly at random and explicitly compute $\min_{b\in B} \|a-b\|_1$. In general, we can compute the estimator $\hat{z}$ for $\texttt{CH}(A,B)$ by averaging over s uniformly chosen samples, resulting in runtime $O(nds)$. It is easy to see that the resulting estimator is un-biased, i.e. $\mathbb{E}[\hat{z}] = \texttt{CH}(A,B)$. However, if a small constant fraction of elements in A contribute significantly to $\texttt{CH}(A,B)$, then $s = \Omega(n)$ samples could be necessary to obtain, say, a 1% relative error estimate with constant probability. Since each sample requires a linear scan to find the nearest neighbor, this would result in a quadratic runtime.

While such an approach has good empirical performance for well-behaved datasets, it does not work for data sets where the distribution of the distances from points in A to their nearest neighbors in B is skewed. Further, it is computationally prohibitive to verify the quality of the approximation given by uniform sampling. Towards proving Theorem 3.1.1, it is paramount to obtain an algorithm that works regardless of the structure of the input dataset.

A more nuanced approach is to perform *importance sampling* where we sample $a \in A$ with probability proportional to its contribution to $\texttt{CH}(A,B)$. In particular, if we had access to a distribution, $\boldsymbol{D}_a$, over elements $a \in A$ such that, $\min_{b\in B}\|a-b\|_1 \le \boldsymbol{D}_a \le \lambda \min_{b\in B}\|a-b\|_1$, for some parameter $\lambda > 1$, then sampling $O(\lambda)$ samples results in an estimator $\hat{z}$ that is unbiased *and* within 1% relative error to the true answer with probability at least 99%. Formally, we consider the estimator defined in Algorithm 1, where we assume access to $\texttt{CrudeNN}(A,B)$, a sub-routine which receives as input A and B and outputs estimates $\boldsymbol{D}_a \in \mathbb{R}_{\ge 0}$ for each $a \in A$ which is guaranteed to be an upper bound for $\min_{b\in B}\|a-b\|_1$. Based on the values $\{\boldsymbol{D}_a\}_{a\in A}$ we construct an importance sampling distribution $\mathcal{D}$ supported on A. Our algorithm samples points from the distribution $\mathcal{D}$, exactly calculates the contribution to the Chamfer distance of only the sampled points, and then appropriately re-weights these values to form the final estimate.

Thus obtain the following lemma which bounds the performance of our estimator.

Lemma 3.1.2 (Variance Bounds for Chamfer Estimate)**.** *Let $n, d \in \mathbb{N}$ and suppose A, B are two subsets of $\mathbb{R}^d$ of size at most n. For any $T \in \mathbb{N}$, the output $\boldsymbol{\eta}$ of* $\texttt{Chamfer-Estimate}(A,B,T)$ *satisfies*

$$\mathbb{E}[\boldsymbol{\eta}] = \texttt{CH}(A,B),$$
$$Var[\boldsymbol{\eta}] \le \frac{1}{T} \cdot \texttt{CH}(A,B)^2 \left(\frac{\boldsymbol{D}}{\texttt{CH}(A,B)} - 1 \right),$$

Algorithm 1 The `Chamfer-Estimate` Algorithm.

```
1: Input: Two subsets $A, B \subset \mathbb{R}^d$ of size at most $n$, and a parameter $T \in \mathbb{N}$.
2: Output: A number $\boldsymbol{\eta} \in \mathbb{R}_{\geq 0}$.
3: procedure CHAMFER-ESTIMATE($(A, B, T)$)
4:     Execute the algorithm CrudeNN($A, B$), and let the output be a set of positive real
       numbers $\{\boldsymbol{D}_a\}_{a \in A}$ which always satisfy $\boldsymbol{D}_a \geq \min_{b \in B} \|a - b\|_1$. Let $\boldsymbol{D} := \sum_{a \in A} \boldsymbol{D}_a$.
5:     Construct the probability distribution $\mathcal{D}$, supported on the set $A$, which satisfies that
       for every $a \in A$,
           $$\Pr_{\boldsymbol{x} \sim \mathcal{D}}[\boldsymbol{x} = a] := \frac{\boldsymbol{D}_a}{\boldsymbol{D}}.$$
6:     For $\ell \in [T]$, sample $\boldsymbol{x}_\ell \sim \mathcal{D}$ and spend $O(|B|d)$ time to compute
           $$\boldsymbol{\eta}_\ell := \frac{\boldsymbol{D}}{\boldsymbol{D}_{\boldsymbol{x}_\ell}} \cdot \min_{b \in B} \|\boldsymbol{x}_\ell - b\|_1.$$
7:     Output
           $$\boldsymbol{\eta} := \frac{1}{T} \sum_{\ell=1}^{T} \boldsymbol{\eta}_\ell.$$
8: end procedure
```

for $\boldsymbol{D}$ in Algorithm 1. The expectations and variance are over the randomness in the samples of Line 6 of `Chamfer-Estimate`(A, B, T)*. In particular,*

$$\Pr\left[|\boldsymbol{\eta} - \texttt{CH}(A, B)| \geq \varepsilon \cdot \texttt{CH}(A, B)\right] \leq \frac{1}{\varepsilon^2 \cdot T}\left(\frac{\boldsymbol{D}}{\texttt{CH}(A, B)} - 1\right).$$

The proof of Lemma 3.1.2 follows from a standard analysis of importance sampling and is given in Section 3.2. Observe, if $\boldsymbol{D} \leq \lambda \texttt{CH}(A, B)$, it suffices to sample $T = O(\lambda/\varepsilon^2)$ points in A, leading to a running time of $O(nd\lambda/\varepsilon^2)$.

Obtaining importance sampling probabilities. It remains to show how to implement the `CrudeNN`(A, B) subroutine to obtain the distribution over elements in A which is a reasonable over-estimator of the true probabilities. A natural first step is to consider performing an $O(\log n)$-approximate nearest neighbor search (NNS): for every $a' \in A$, find $b' \in B$ satisfying $\|a' - b'\|_1 / \min_{b \in B} \|a' - b\|_1 = O(\log n)$. This leads to the desired guarantees on $\{\boldsymbol{D}_a\}_{a \in A}$. Unfortunately, the state of the art algorithms for $O(\log n)$-approximate NNS, even under the ℓ_1 norm, posses extraneous poly$(\log n)$ factors in the runtime, resulting in a significantly higher running time. These factors are even higher for the ℓ_2 norm. Therefore, instead of performing a direct reduction to approximate NNS, we open up the approximate NNS black-box and give a simple algorithm which directly satisfies our desired guarantees on $\{\boldsymbol{D}_a\}_{a \in A}$.

Algorithm 2 The `CrudeNN` Algorithm.

```
1: Input: Two subsets $A, B$ of a metric space $(X, \|\cdot\|_1)$ of size at most $n$ such that all
   non-zero distances between any point in $A$ and any point in $B$ is between 1 and poly$(n/\varepsilon)$.
   We assume access to a locality-sensitive hash family at every scale $\mathcal{H}(r)$ for any $r \geq 0$
   satisfying conditions of Definition 3.2.1. (We show in Section 3.2 that, for $\ell_1$ and $\ell_2$, the
   desired hash families exist, and that distances between 1 and poly$(n/\varepsilon)$ is without loss
   of generality).
2: Output: A list of numbers $\{\boldsymbol{D}_a\}_{a \in A}$ where $\boldsymbol{D}_a \geq \min_{b \in B} \|a - b\|_1$.
3: procedure CRUDENN($(A, B)$)
4:     We instantiate $L = O(\log(n/\varepsilon))$ and for $i \in \{0, \dots, L\}$, we let $r_i = 2^i$.
5:     For each $i \in \{0, \dots, L\}$ sample a hash function $\boldsymbol{h}_i \colon X \to U$ from $\boldsymbol{h}_i \sim \mathcal{H}(r_i)$.
6:     For each $a \in A$, find the smallest $i \in \{0, \dots, L\}$ for which there exists a point $b \in B$
   with $\boldsymbol{h}_i(a) = \boldsymbol{h}_i(b)$, and set $\boldsymbol{D}_a = \|a - b\|_1$.
         • The above may be done by first hashing each point $b \in B$ and $i \in \{0, \dots, L\}$
           according to $\boldsymbol{h}_i(b)$. Then, for each $a \in A$, we iterate through $i \in \{0, \dots, L\}$ while
           hashing $a$ according to $\boldsymbol{h}_i(a)$ until the first $b \in B$ with $\boldsymbol{h}_i(a) = \boldsymbol{h}_i(b)$ is found.
7: end procedure
```

To begin with, we assume that the aspect ratio of all pair-wise distances is bounded by a fixed polynomial, poly(n/ε) (we defer the reduction from an arbitrary input to one with polynomially bounded aspect ratio to Lemma 3.2.2). We proceed via computing $O(\log(n/\varepsilon)$ different (randomized) partitions of the dataset $A \cup B$. The i-th partition, for $1 \leq i \leq O(\log(n/\varepsilon))$, can be written as $A \cup B = \cup_j \mathcal{P}_j^i$ and approximately satisfies the property that points in $A \cup B$ that are at distance at most 2^i will be in the same partition $\mathcal{P}_j^i$ with sufficiently large probability. To obtain these components, we use a family of *locality-sensitive hash functions*, whose formal properties are given in Definition 3.2.1. Intuitively, these hash functions guarantee that:

1. For each $a' \in A$, its *true* nearest neighbor $b' \in B$ falls into the *same* component as a' in the i_0-th partition, where $2^{i_0} = \Theta(\|a' - b'\|_1)$ [1], and

2. Every other extraneous $b \neq b'$ is *not* in the same component as a' for each $i < i_0$.

It is easy to check that any hash function that satisfies the aforementioned guarantees yields a valid set of distances $\{\boldsymbol{D}_a\}_{a \in A}$ as follows: for every $a' \in A$, find the smallest i_0 for which there exists a $b' \in B$ in the same component as a' in the i_0-th partition. Then set $\boldsymbol{D}_{a'} = \|a' - b'\|_1$. Intuitively, the b' we find for any fixed a' in this procedure will have distance that is at least the closest neighbor in B and with good probability, it won't be too much larger. A caveat here is that we cannot show the above guarantee holds for $2^{i_0} = \Theta(\|a' - b'\|_1)$. Instead, we obtain the slightly weaker guarantee that, *in the expectation*, the partition b'

[1]Recall we assumed all distances are between 1 and poly(n) resulting in only $O(\log n)$ different partitions

lands in is a $O(\log n)$-approximation to the minimum distance, i.e. $2^{i_0} = \Theta(\log n \cdot \|a' - b'\|_1)$. Therefore, after running `CrudeNN`(A, B), setting $\lambda = \log n$ suffices for our $O\left(nd \log(n)/\varepsilon^2\right)$ time algorithm. We formalize this argument in the following lemma:

Lemma 3.1.3 (Oversampling with bounded Aspect Ratio)**.** *Let (X, d_X) be a metric space with a locality-sensitive hash family at every scale (see Definition 3.2.1). Consider two subsets $A, B \subset X$ of size at most n and any $\varepsilon \in (0, 1)$ satisfying*

$$1 \leq \min_{\substack{a \in A, b \in B \\ a \neq b}} d_X(a, b) \leq \max_{a \in A, b \in B} d_X(a, b) \leq \text{poly}(n/\varepsilon).$$

Algorithm 2, `CrudeNN`(A, B)*, outputs a list of (random) positive numbers $\{\boldsymbol{D}_a\}_{a \in A}$ which satisfy the following two guarantees:*

- *With probability 1, every $a \in A$ satisfies $\boldsymbol{D}_a \geq \min_{b \in B} d_X(a, b)$.*
- *For every $a \in A$, $\mathbb{E}[\boldsymbol{D}_a] \leq O(\log n) \cdot \min_{b \in B} d_X(a, b)$.*

Further, Algorithm 2, runs in time $O(dn \log(n/\varepsilon))$ time, assuming that each function used in the algorithm can be evaluated in $O(d)$ time.

Proof Sketch for Theorem 3.1.1. Given the lemmas above, it is straight-forward to complete the proof of Theorem 3.1.1. First, we reduce to the setting where the aspect ratio is $\text{poly}(n/\varepsilon)$ (see Lemma 3.2.2 for a formal reduction). We then invoke Lemma 3.1.3 and apply Markov's inequality to obtain a set of distances $\boldsymbol{D}_a$ such that with probability at least 99/100, for each $a \in A$, $\min_{b \in B} \|a - b\|_1 \leq \boldsymbol{D}_a$ and $\sum_{a \in A} \boldsymbol{D}_a \leq O(\log(n))\texttt{CH}(A, B)$. We then invoke Lemma 3.1.2 and set the number of samples, $T = O\left(\log(n)/\varepsilon^2\right)$. The running time of our algorithm is then given by the time of `CrudeNN`(A, B), which is $O(nd \log(n/\varepsilon))$, and the time needed to evaluate the estimator in Lemma 3.1.2, requiring $O\left(nd \log(n)/\varepsilon^2\right)$ time. Refer to Section 3.2 for the full proof. $\square$

Other Related Works We note that importance sampling is a popular technique used for speeding up geometric algorithms. For example, [106] uses it to obtain a fast c-approximate algorithm for computing Earth Mover Distance (EMD) in two (or any constant) dimensions, for some constant $c > 2$. However, the application and implementation of importance sampling in that paper is quite different from ours.

In [106], the space containing all input points is subdivided into regions, and the total EMD value is represented as a sum of EMDs restricted to point-sets in each region (plus an additional representing the "global" EMD). The EMD cost in each region is then approximated quickly by embedding EMD into ℓ_1 using a randomly shifted quadtree with logarithmic distortion; these estimations define the sampling probabilities.

In contrast, in our work, the value of the Chamfer distance is exactly equal to the sum of distances from each point to its nearest neighbor, so there is no decomposition involved. Instead, we approximate each distance to nearest neighbor using a randomized hierarchical decomposition; for the case of the ℓ_1 norm, each level of the decomposition partitions the space into rectangular boxes, as in quadtrees. Crucially, however, to ensure that the running time of our algorithm is within the stated bounds, we cannot use a standard randomly shifted quadtree where each level is shifted by the same random vector (as in [106]). This is because shifting all levels by the same amount only ensures that the expected distortion between a fixed pair of points is logarithmic; to ensure that the distance to the nearest neighbor is distorted by $O(\log n)$, we would need to use $O(\log n)$ independent quadtrees and apply the union bound. Instead, we use independent random partitions at each level, and show (Lemma 3.1.3) that this suffices to bound the expected distortion of the distance to the nearest neighbor, without incurring any additional factors. This makes it possible to obtain the running time as stated.

There are other works on quickly computing EMD and related distances. For example, the algorithm of [12] runs in time that is linear in the number of distances, i.e. it runs in $\Omega(n^2)$ time and gets a $1+\varepsilon$ approximation to EMD. This means that their approach requires a runtime quadratic in the size of the dataset n. In contrast, Chamfer distance admits a trivial time $O(n^2)$ algorithm and our main contribution is to provide a nearly linear $O(n\log(n)/\varepsilon^2)$ algorithm to get $(1+\varepsilon)$-approximation to Chamfer distance.

3.2 Full Analysis of the Upper Bound

Proof of Lemma 3.1.2. The proof follows from a standard analysis of importance sampling. The fact that our estimator $\boldsymbol{\eta}$ is unbiased holds from the definition of $\boldsymbol{\eta}_\ell$, since we are re-weighting samples according to the probability with which they are sampled in $\mathcal{D}$ (in particular, the estimator is unbiased for all distributions $\mathcal{D}$ where $\boldsymbol{D}_a > 0$ for all $a \in A$). The bound on the variance is then a simple calculation:

$$\begin{aligned}
\mathrm{Var}\left[\boldsymbol{\eta}\right] &\le \frac{1}{T}\cdot\left(\left[\sum_{a\in A}\left(\frac{\boldsymbol{D}}{\boldsymbol{D}_a}\right)\min_{b\in B}\|a-b\|_2^2\right] - \mathtt{CH}(A,B)^2\right)\\
&\le \frac{1}{T}\cdot\left[\sum_{a\in A}\min_{b\in B}\|a-b\|_2\cdot\boldsymbol{D}\right] - \frac{\mathtt{CH}(A,B)^2}{T}\\
&\le \frac{1}{T}\cdot\mathtt{CH}(A,B)^2\left(\frac{\boldsymbol{D}}{\mathtt{CH}(A,B)}-1\right).
\end{aligned}$$

The final probability bound follows from Chebyshev's inequality. □

Locality Sensitive Hashing at every scale. We now discuss how to find such partitions. For the ℓ_1 distance, each partition i is formed by imposing a (randomly shifted) grid of side length 2^i on the dataset. Note that while the grid partitions the entire space $\mathbb{R}^d$ into infinitely many components, we can efficiently enumerate over the *non empty* components which actually contain points in our dataset. To this end, we introduce the following definition:

Definition 3.2.1 (Hashing at every scale)**.** *There exists a fixed constant $c_1 > 0$ and a parameterized family $\mathcal{H}(r)$ of functions from X to some universe U such that for all $r > 0$, and for every $x, y \in X$*

1. *Close points collide frequently:*

$$\Pr_{\boldsymbol{h} \sim \mathcal{H}(r)} [\boldsymbol{h}(x) \neq \boldsymbol{h}(y)] \leq \frac{\|x - y\|_1}{r},$$

2. *Far points collide infrequently:*

$$\Pr_{\boldsymbol{h} \sim \mathcal{H}(r)} [\boldsymbol{h}(x) = \boldsymbol{h}(y)] \leq \exp\left(-c_1 \cdot \frac{\|x - y\|_1}{r}\right).$$

We are now ready to make this approach concrete via the following lemma:

Lemma 3.2.1 (Oversampling with bounded Aspect Ratio)**.** *Let (X, d_X) be a metric space with a locality-sensitive hash family at every scale (see Definition 3.2.1). Consider two subsets $A, B \subset X$ of size at most n and $\varepsilon \in (0,1)$ satisfying*

$$1 \leq \min_{\substack{a \in A, b \in B \\ a \neq b}} d_X(a,b) \leq \max_{a \in A, b \in B} d_X(a,b) \leq \mathrm{poly}(n/\varepsilon).$$

Algorithm 2, `CrudeNN`(A, B), outputs a list of (random) positive numbers $\{\boldsymbol{D}_a\}_{a \in A}$ which satisfy the following two guarantees:

- *With probability 1, every $a \in A$ satisfies $\boldsymbol{D}_a \geq \min_{b \in B} d_X(a,b)$.*
- *For every $a \in A$, $\mathbb{E}[\boldsymbol{D}_a] \leq O(\log n) \cdot \min_{b \in B} d_X(a,b)$.*

Further, Algorithm 2, runs in time $O(nd \log(n/\varepsilon))$ time, assuming that each function used in the algorithm can be evaluated in $O(d)$ time.

Finally, we show that it always suffices to assume bounded aspect ratio:

Lemma 3.2.2 (Reduction to bounded Aspect Ratio)**.** *Given an instance $A, B \subset \mathbb{R}^d$ such that $|A|, |B| \leq n$, and $0 < \varepsilon < 1$ there exists an algorithm that runs in time $O(nd \log(n)/\varepsilon^2)$ and outputs a partition $A_1, A_2, \ldots A_T$ of A and $B_1, B_2, \ldots B_T$ of B such that $T = O(n)$ and for each $t \in [T]$,*

$$1 \leq \min_{\substack{a \in A_t, b \in B_t \\ a \neq b}} \|a - b\|_1 \leq \max_{a \in A_t, b \in B_t} \|a - b\|_1 \leq \mathrm{poly}(n/\varepsilon).$$

Further,

$$(1 - \varepsilon)\mathtt{CH}(A, B) \leq \sum_{t \in [T]} \mathtt{CH}(A_t, B_t) \leq (1 + \varepsilon)\mathtt{CH}(A, B).$$

We defer the proofs of Lemma 3.2.1 and Lemma 3.2.2 to sub-sections 3.2.1 and 3.2.3 respectively. We are now ready to complete the proof of Theorem 3.1.1.

Proof of Theorem 3.1.1. Observe, by Lemma 3.2.2, we can partition the input into pairs $(A_t, B_t)_{t\in[T]}$ such that each pair has aspect ratio at most $\mathrm{poly}(n/\varepsilon)$ and the $\mathtt{CH}(A,B)$ is well-approximated by the direct sum of $\mathtt{CH}(A_t, B_t)$. Next, repeating the construction from Lemma 3.2.1, and applying Markov's inequality, we have list $\{\boldsymbol{D}_a\}_{a\in A}$ such that with probability at least 99/100, for all $a \in A$, $\boldsymbol{D}_a \geq \min_{b\in B} \|a-b\|_1$ and $\boldsymbol{D} = \sum_{a\in A} \boldsymbol{D}_a \leq O(\log(n))\mathtt{CH}(A,B)$. Invoking Lemma 3.1.2 with the aforementioned parameters, and $T = O\left(\log(n)/\varepsilon^2\right)$ suffices to obtain an estimator η which is a $(1\pm\varepsilon)$ relative-error approximation to $\mathtt{CH}(A,B)$. Since we require computing the exact nearest neighbor for at most $O\left(\log(n)/\varepsilon^2\right)$ points, the running time is dominated by $O\left(nd\log(n)/\varepsilon^2\right)$, which completes the proof. □

3.2.1 Analysis for CrudeNN

In this subsection, we focus analyze the `CrudeNN` algorithm and provide a proof for Lemma 3.2.1. A construction of hash family satisfying Definition 3.2.1 is given in Section 3.2.2. Each function from the family can be evaluated in $O(d)$ time per point. We are now ready to prove Lemma 3.2.1.

Proof of Lemma 3.2.1. We note that the first item is trivially true, since $\mathtt{CrudeNN}(A,B)$ always sets $\boldsymbol{D}_a$ to be some distance between a and a point in B. Thus, this distance can only be larger than the true minimum distance. The more challenging aspect is obtaining an upper bound on the expected value of $\boldsymbol{D}_a$. Consider a fixed setting of $a \in A$, and the following setting of parameters:

$$b = \arg\min_{b'\in B} d_X(a,b') \qquad \gamma_a = d_X(a,b) \qquad i_0 = \lceil \log_2 \gamma_a \rceil,$$

and notice that since γ_a is between 1 and $\mathrm{poly}(n/\varepsilon)$, we have i_0 is at least 0 and at most $L = O(\log(n/\varepsilon))$. We will upper bound the expectation of $\boldsymbol{D}_a$ by considering a parameter $c > 1$ (which will later be set to $O(\log n)$), and integrating over the probability that $\boldsymbol{D}_a$ is at least γ, for all $\gamma \geq c\cdot\gamma_a$:

$$\mathbb{E}\left[\boldsymbol{D}_a\right] \leq c\cdot\gamma_a + \int_{c\gamma_a}^{\infty} \Pr\left[\boldsymbol{D}_a \geq \gamma\right] d\gamma. \tag{3.1}$$

We now show that for any $a \in A$, the probability that $\boldsymbol{D}_a$ is larger than γ can be appropriately bounded. Consider the following two bad events.

- $\boldsymbol{E}_1(\gamma)$: This event occurs when there exists a point $b' \in B$ at distance at least γ from a and there exists an index $i \leq i_0$ for which $\boldsymbol{h}_i(a) = \boldsymbol{h}_i(b')$.
- $\boldsymbol{E}_2(\gamma)$: This event occurs when there exists an index $i > i_0$ such that:
  - For every $i' \in \{i_0, \ldots, i-1\}$, we have $\boldsymbol{h}_{i'}(a) \neq \boldsymbol{h}_{i'}(b)$ for all $b \in B$.

– There exists $b' \in B$ at distance at least γ from a where $\boldsymbol{h}_i(a) = \boldsymbol{h}_i(b')$.

We note that whenever $\texttt{CrudeNN}(A, B)$ set $\boldsymbol{D}_a$ larger than γ, one of the two events, $\boldsymbol{E}_1(\gamma)$ or $\boldsymbol{E}_2(\gamma)$, must have been triggered. To see why, suppose $\texttt{CrudeNN}(A, B)$ set $\boldsymbol{D}_a$ to be larger than γ because a point $b' \in B$ with $d_X(a, b') \geq \gamma$ happened to have $\boldsymbol{h}_i(a) = \boldsymbol{h}_i(b')$, for an index $i \in \{0, \dots, L\}$, and that the index i was the first case where it happened. If $i \leq i_0$, this is event $\boldsymbol{E}_1(\gamma)$. If $i > i_0$, we claim event $\boldsymbol{E}_2(\gamma)$ occurred: in addition to $\boldsymbol{h}_i(a) = \boldsymbol{h}_i(b')$, it must have been the case that, for all $i' \in \{i_0, \dots, i-1\}$, $\boldsymbol{h}_{i'}(a) \neq \boldsymbol{h}_{i'}(b)$ (otherwise, i would not be the first index). We will upper bound the probability that either event $\boldsymbol{E}_1(\gamma)$ or $\boldsymbol{E}_2(\gamma)$ occurs. We make use of the tail bounds as stated in Definition 3.2.1. The upper bound for the probability that $\boldsymbol{E}_1(\gamma)$ is simple, since it suffices to union bound over at most n points at distance larger than γ, using the fact that $r_{i_0} = 2^{i_0}$ is at most $2 \cdot \gamma_a$:

$$\Pr\left[\boldsymbol{E}_1(\gamma)\right] \leq n \cdot \exp\left(-c_1 \cdot \frac{\gamma}{2\gamma_a}\right). \tag{3.2}$$

We will upper bound the probability that event $\boldsymbol{E}_2(\gamma)$ a bit more carefully. We will use the fact that for all i, the parameter r_i is always between $2^{i-i_0}\gamma_a$ and $2^{i-i_0+1}\gamma_a$.

$$\begin{aligned}
\Pr\left[\boldsymbol{E}_2(\gamma)\right] &\leq \sum_{i > i_0} \left(\prod_{i'=i_0}^{i-1} \frac{\gamma_a}{r_{i'}}\right) \cdot \max\left\{n \cdot \exp\left(-c_1 \cdot \frac{\gamma}{r_i}\right), 1\right\} \\
&\leq \sum_{i > i_0} 2^{-(0 + \dots + (i-1-i_0))} \max\left\{\exp\left(\ln(n) - c_1 \cdot \frac{\gamma}{2^{i-i_0+1} \cdot \gamma_a}\right), 1\right\} \\
&\leq \sum_{k \geq 0} 2^{-\Omega(k^2)} \cdot \max\left\{\exp\left(\ln(n) - c_1 \cdot \frac{\gamma}{2^{k+2} \cdot \gamma_a}\right), 1\right\}.
\end{aligned} \tag{3.3}$$

With the above two upper bounds in place, we upper bound (3.1) by dividing the integral into the two contributing summands, from $\boldsymbol{E}_1(\gamma)$ and $\boldsymbol{E}_2(\gamma)$, and then upper bounding each individually. Namely, we have

$$\int_{\gamma : c\gamma_a}^{\infty} \Pr\left[\boldsymbol{D}_a \geq \gamma\right] d\gamma \leq \int_{\gamma : c\gamma_a}^{\infty} \Pr\left[\boldsymbol{E}_1(\gamma)\right] d\gamma + \int_{\gamma : c\gamma_a}^{\infty} \Pr\left[\boldsymbol{E}_2(\gamma)\right] d\gamma.$$

The first summand can be simply upper bounded by using the upper bound from (3.2), where we have

$$\int_{\gamma : c\gamma_a}^{\infty} \Pr\left[\boldsymbol{E}_1(\gamma)\right] d\gamma \leq \int_{\gamma : c\gamma_a}^{\infty} n \exp\left(-c_1 \cdot \frac{\gamma}{2\gamma_a}\right) d\gamma \leq \frac{n \cdot 2\gamma_a}{c_1} \cdot e^{-c_1 c/2} \leq \gamma_a$$

for a large enough $c = \Theta(\log n)$. The second summand is upper bounded by the upper bound in (3.3), while being slightly more careful in the computation. In particular, we first commute the summation over k and the integral; then, for each $k \geq 0$, we define

$$\alpha_k := 2^{k+3} \ln(n) \cdot \gamma_a / c_1,$$

and we break up the integral into the interval $[c \cdot \gamma_a, \alpha_k]$ (if $\alpha_k < c\gamma_a$, the interval is empty), as well as $[\alpha_k, \infty)$:

$$\begin{aligned}\int_{\gamma : c\gamma_a}^{\infty} \Pr\left[\boldsymbol{E}_2(\gamma)\right] d\gamma &\leq \sum_{k \geq 0} 2^{-\Omega(k^2)} \int_{\gamma : c\gamma_a}^{\infty} \max\left\{\exp\left(\ln(n) - c_1 \cdot \frac{\gamma}{2^{k+2} \cdot \gamma_a}\right), 1\right\} d\gamma \\ &\leq \sum_{k \geq 0} 2^{-\Omega(k^2)} \left((\alpha_k - c \cdot \gamma_a)^+ + \int_{\gamma : \alpha_k}^{\infty} \exp\left(-\frac{c_1}{2} \cdot \frac{\gamma}{2^{k+2}\gamma_a}\right) d\gamma\right),\end{aligned}$$

where in the second inequality, we used the fact that the setting of α_k, the additional $\ln(n)$ factor in the exponent can be removed up to a factor of two. Thus,

$$\int_{\gamma : c\gamma_a}^{\infty} \Pr\left[\boldsymbol{E}_2(\gamma)\right] d\gamma \leq \sum_{k \geq 0} 2^{-\Omega(k^2)} \left(\gamma_a \cdot O(2^k \log n) + \gamma_a \cdot O(2^k)\right) = O(\log n) \cdot \gamma_a.$$

Finally, the running time is dominated by the cost of evaluating $O(\log(n/\varepsilon))$ functions on n points in dimension d. Since each evaluation takes $O(d)$ time, the bound follows. □

3.2.2 Locality-Sensitive Hashing at Every Scale

Lemma 3.2.3 (Constructing a LSH at every scale). *For any $r \geq 0$ and any $d \in \mathbb{N}$, there exists a hash family $\mathcal{H}(r)$ such that for any two points $x, y \in \mathbb{R}^d$,*

$$\Pr_{\boldsymbol{h} \sim \mathcal{H}(r)}\left[\boldsymbol{h}(x) \neq \boldsymbol{h}(y)\right] \leq \frac{\|x - y\|_1}{r}$$

$$\Pr_{\boldsymbol{h} \sim \mathcal{H}(r)}\left[\boldsymbol{h}(x) = \boldsymbol{h}(y)\right] \leq \exp\left(-\frac{\|x - y\|_1}{r}\right).$$

In addition, for any $\boldsymbol{h} \sim \mathcal{H}(r)$, $\boldsymbol{h}(x)$ may be computed in $O(d)$ time.

Proof. The construction proceeds in the following way: in order to generate a function $\boldsymbol{h} \colon \mathbb{R}^d \to \mathbb{Z}^d$ sampled from $\mathcal{H}(r)$,

- We sample a random vector $\boldsymbol{z} \sim [0, r]^d$.
- We let
$$\boldsymbol{h}(x) = \left(\left\lceil \frac{x_1 + \boldsymbol{z}_1}{r} \right\rceil, \left\lceil \frac{x_2 + \boldsymbol{z}_2}{r} \right\rceil, \dots, \left\lceil \frac{x_d + \boldsymbol{z}_d}{r} \right\rceil\right).$$

Fix $x, y \in \mathbb{R}^d$. If $\boldsymbol{h}(x) \neq \boldsymbol{h}(y)$, there exists some coordinate $k \in [d]$ on which $\boldsymbol{h}(x)_k \neq \boldsymbol{h}(y)_k$. This occurs whenever (i) $|x_k - y_k| > r$, or (ii) $|x_k - y_k| \leq r$, but $\boldsymbol{z}_k$ happens to fall within an interval of length $|x_k - y_k|$, thereby separating x from y. By a union bound,

$$\Pr_{\boldsymbol{h} \sim \mathcal{H}(r)}\left[\boldsymbol{h}(x) \neq \boldsymbol{h}(y)\right] \leq \sum_{k=1}^{d} \frac{|x_k - y_k|}{r} = \frac{\|x - y\|_1}{r}.$$

On the other hand, in order for $\boldsymbol{h}(x) = \boldsymbol{h}(y)$, it must be the case that every $|x_k - y_k| \leq r$, and in addition, the threshold $\boldsymbol{z}_k$ always avoids an interval of length $|x_k - y_k|$. The probability that this occurs is

$$\Pr_{\boldsymbol{h} \sim \mathcal{H}(r)} [\boldsymbol{h}(x) = \boldsymbol{h}(y)] = \prod_{k=1}^{d} \max\left\{0, 1 - \frac{|x_k - y_k|}{r}\right\} \leq \exp\left(-\sum_{k=1}^{d} \frac{|x_k - y_k|}{r}\right)$$
$$\leq \exp\left(-\frac{\|x-y\|_1}{r}\right).$$

□

Extending the above construction to ℓ_2 follows from embedding the points $A \cup B$ into ℓ_1 via a standard construction given in Theorem 2.0.2.

The map $\boldsymbol{T} \colon \mathbb{R}^d \to \mathbb{R}^k$ from Theorem 2.0.2 with $k = O\left(\log n/\varepsilon^2\right)$ gives an embedding of $A \cup B$ into ℓ_1^k of distortion $(1 \pm \varepsilon)$ with high probability. Formally, with probability at least $1 - 1/n$ over the draw of $\boldsymbol{T}$ with $t = O\left(\log n/\varepsilon^2\right)$, every $a \in A$ and $b \in B$ satisfies

$$(1-\varepsilon)\|a-b\|_2 \leq \|\boldsymbol{T}(a) - \boldsymbol{T}(b)\|_1 \leq (1+\varepsilon)\|a-b\|_2.$$

This embedding has the effect of reducing ℓ_2 to ℓ_1 without affecting the Chamfer distance of the mapped points by more than a $(1 \pm \varepsilon)$-factor. In addition, the embedding incurs an extra additive factor of $O\left(nd \log n/\varepsilon^2\right)$ to the running time in order to perform the embedding for all points.

3.2.3 Reduction to $\mathrm{poly}(n/\varepsilon)$ Aspect Ratio for ℓ_p, $p \in [1, 2]$

In this section, we discuss how to reduce to the case of a $\mathrm{poly}(n/\varepsilon)$ aspect ratio. The reduction proceeds by first obtaining a very crude estimate of $\mathtt{CH}(A, B)$ (which will be a $\mathrm{poly}(n)$-approximation), applying a locality-sensitive hash function in order to partition points of A and B which are significantly farther than $\mathrm{poly}(n) \cdot \mathtt{CH}(A, B)$. Finally, we add $O(\log n)$ coordinates and add random vector of length $\mathrm{poly}(\varepsilon/n) \cdot \mathtt{CH}(A, B)$ in order to guarantee that the minimum distance is at least $\mathrm{poly}(\varepsilon/n) \cdot \mathtt{CH}(A, B)$ without changing $\mathtt{CH}(A, B)$ significantly.

Proof of Lemma 3.2.2. **Partitioning Given Very Crude Estimates.** In particular, suppose that with an $O(nd) + O(n \log n)$ time computation, we can achieve a value of $\boldsymbol{\eta} \in \mathbb{R}_{\geq 0}$ which satisfies

$$\mathtt{CH}(A, B) \leq \boldsymbol{\eta} \leq c \cdot \mathtt{CH}(A, B),$$

with high probability (which we will show how to do briefly with $c = \mathrm{poly}(n)$). Then, consider sampling $\boldsymbol{h} \sim \mathcal{H}(cn \cdot \boldsymbol{\eta})$ and partitioning A and B into equivalence classes according to where they hash to under $\boldsymbol{h}$. The probability that two points at distance farther than $O(\log n) \cdot cn \cdot \boldsymbol{\eta}$ collide under $\boldsymbol{h}$ is small enough to union bound over at most n^2 many possible pairs of vectors. In addition, the probability that there exists $a \in A$ for which $b \in B$ minimizing $\|a - b\|_p$

satisfies $\boldsymbol{h}(a) \neq \boldsymbol{h}(b)$ is at most $\texttt{CH}(A, B)/(cn \cdot \boldsymbol{\eta}) \leq 1/n$. This latter inequality implies that computing the Chamfer distance of the corresponding parts in the partition and summing them is equivalent to computing $\texttt{CH}(A, B)$.

Getting Very Crude Estimates. We now show how to obtain a poly(n)-approximation to $\texttt{CH}(A, B)$ in time $O(nd) + O(n \log n)$ for points in $\mathbb{R}^d$ with ℓ_p distance. This is done via the p-stable sketch of Indyk [105]. In particular, we sample a vector $\boldsymbol{g} \in \mathbb{R}^d$ by independent p-stable random variables (for instance, $\boldsymbol{g}$ is a standard Gaussian vector for $p = 2$ and a vector of independent Cauchy random variables for $p = 1$). We may then compute the scalar random variables $\{\langle a, \boldsymbol{g}\rangle\}_{a\in A}$ and $\{\langle b, \boldsymbol{g}\rangle\}_{b\in B}$, which give a projection onto a one-dimensional space. By p-stability, for any $a \in A$ and $b \in B$, the distribution of $\langle a, \boldsymbol{g}\rangle - \langle b, \boldsymbol{g}\rangle$ is exactly as $\|a - b\|_p \cdot \boldsymbol{g}'$, where $\boldsymbol{g}'$ is an independent p-stable random variable. Hence, we will have that for every $a \in A$ and $b \in B$,

$$\frac{\|a-b\|_p}{\operatorname{poly}(n)} \leq |\langle a, \boldsymbol{g}\rangle - \langle b, \boldsymbol{g}\rangle| \leq \|a-b\|_p \cdot \operatorname{poly}(n),$$

with probability $1 - 1/\operatorname{poly}(n)$ and hence $\texttt{CH}(\{\langle a, \boldsymbol{g}\rangle\}_{a\in A}, \{\langle b, \boldsymbol{g}\rangle\}_{b\in B})$, which is computable by 1-dimensional nearest neighbor search (i.e., repeatedly querying a binary search tree), gives a poly(n)-approximation to $\texttt{CH}(A, B)$.

Adding Distance Finally, we now note that $\boldsymbol{\eta}/c$ gives us a lower bound on $\texttt{CH}(A, B)$. Suppose we append $O(\log n)$ coordinates to each point and in those coordinates, we add a random vector of norm $\varepsilon \cdot \boldsymbol{\eta}/(cn)$. With high probability, every pair of points is now at distance at least $\varepsilon \cdot \boldsymbol{\eta}/(cn)$. In addition, the Chamfer distance between the new set of points increases by at most an additive $O(\varepsilon\boldsymbol{\eta}/c)$, which is at most $O(\varepsilon) \cdot \texttt{CH}(A, B)$, proving Lemma 3.2.2.

□

3.3 Lower Bound for Reporting the Alignment

We presented an algorithm that, in time $O(nd \log(n)/\varepsilon^2)$, produces a $(1 + \varepsilon)$-approximation to $\texttt{CH}(A, B)$. It is natural to ask whether it is also possible to *report* a mapping $g : A \to B$ whose cost $\sum_{a\in A} \|a - g(a)\|_1$ is within a factor of $1 + \varepsilon$ from $\texttt{CH}(A, B)$. (Our algorithm uses on random sampling and thus does not give such a mapping). This section shows that, under a popular complexity-theoretic conjecture called the *Hitting Set Conjecture* [198], such an algorithm does not exists. For simplicity, we focus on the case when the underlying metric d_X is induced by the Manhattan distance, i.e., $d_X(a, b) = \|a - b\|_1$. The argument is similar for the Euclidean distance, Euclidean distance squared, etc. To state our result formally, we first recall the Hitting Set (HS) problem.

Definition 2.0.1. (Hitting Set (HS) problem) *The input to the problem consists of two sets of vectors $A, B \subseteq \{0, 1\}^d$, and the goal is to determine whether there exists some $a \in A$ such that $a \cdot b \neq 0$ for every $b \in B$. If such an $a \in A$ exists, we say that a* hits *B.*

It is easy to see that the Hitting Set problem can be solved in time $O(n^2 d)$. The Hitting Set Conjecture [198] postulates that this running time is close to the optimal. We recall the hitting set conjecture.

Conjecture 2.0.1. *Suppose $d = \Theta(\log^2 n)$. Then for every constant $\alpha > 0$, no randomized algorithm can solve the Hitting Set problem in $O(n^{2-\alpha})$ time.*

Our result can be now phrased as follows.

Theorem 3.3.1 (Hardness for reporting a mapping)**.** *Let $T(N, D, \varepsilon)$ be the running time of an algorithm ALG that, given sets of $A", B" \subset \{0,1\}^D$ of sizes at most N, reports a mapping $g : A" \to B"$ with cost $(1+\varepsilon)\texttt{CH}(A", B")$, for $D = \Theta(\log^2 N)$ and $\varepsilon = \frac{\Theta(1)}{D}$. Assuming the Hitting Set Conjecture, we have that $T(N, D, \varepsilon)$ is at least $\Omega(N^{2-\delta})$ for any constant $\delta > 0$.*

Proof. To set the notation, we let $n_A = |A|$, $n_B = |B|$.

The proof mimics the argument from [170], which proved a similar hardness result for the problem of computing the Earth-Mover Distance. In particular, Lemma 4.3 from that paper shows the following claim.

Claim 3.3.2. *For any two sets $A, B \subseteq \{0,1\}^d$, there is a mapping $f : \{0,1\}^d \to \{0,1\}^{d"}$ and a vector $v \in \{0,1\}^{d"}$, such that $d" = O(d)$ and for any $a \in A$, $b \in B$:*

- *If $a \cdot b = 0$ then $\|f(a) - f(b)\|_1 = 4d + 2$,*
- *If $a \cdot b > 0$ then $\|f(a) - f(b)\|_1 \geq 4d + 4$,*
- *$\|f(a) - v\|_1 = 4d + 4$.*

Furthermore, each evaluation $f(a)$ can be performed in $O(d)$ time.

We will be running ALG on sets $A" = \{f(a) : a \in A\}$ and $B" = \{f(b) : b \in B\} \cup \{v\}$. It can be seen that, given a reported mapping g, we can assume that for all $a" \in A"$ we have $\|a" - g(a")\|_1 \leq 4d + 4$, as otherwise g can map a'' to v. If for all $a \in A$ there exists $b \in B$ such that $a \cdot b = 0$, i.e., A does not contain a hitting vector, then the optimal mapping cost is $n_A(4d + 2)$. More generally, let H be the set of vectors $a \in A$ hitting B, and let $h = |H|$. It can be seen that

$$\texttt{CH}(A", B") = h(4d + 4) + (n_A - h)(4d + 2) = n_A(4d + 2) + 2h.$$

Thus, if we could compute $\texttt{CH}(A", B")$ exactly, we would determine if $h = 0$ and solve HS. In what follows we show that even an approximate solution can be used to accomplish this task as long as ε is small enough.

Let $t = c\log(n)/\varepsilon$ for some large enough constant $c > 1$. Consider the algorithm $HittingSet(A, B)$ that solves HS by invoking the algorithm ALG.

It can be seen that the first three steps of the algorithm take at most $O(ntd)$ time. Furthermore, if the algorithm terminates, it reports the correct answer, as only vectors a that are guaranteed not to be hitting are removed in the recursion. It remains to bound

Algorithm 3 Reduction from Hitting Set to $(1+\varepsilon)$-approximate CH, implemented using algorithm ALG .

```
1: Input: Two sets $A, B \subset \{0,1\}^d$ of size at most $n$, and an oracle access to $ALG$ that
   computes $(1+\varepsilon)$-approximate CH.
2: Output: Determines whether there exists $a \in A$ such that $a \cdot b > 0$ for all $b \in B$
3: procedure HittingSet($(A, B)$)
4:     Sample (uniformly, without replacement) $\min(t, |A|)$ distinct vectors $a \in A$, and for
       each of them check if $a \cdot b > 0$ for all $B$. If such an $a$ is found, return YES.
5:     Construct $A", B"$ as in Claim 3.3.2, and invoke $ALG$. Let $g : A" \to B"$ be the
       returned map.
6:     Identify the set $M$ containing all $a \in A$ such that $\|g(f(a)) - f(a)\|_1 = 4d + 2$. Note
       that $a \cdot b = 0$ for $b \in B$ such that $f(b) = g(f(a))$.
7:     Recursively execute HittingSet($A - M, B$)
8: end procedure
```

the total number and cost of the recursive steps. To this end, we will show that, with high probability, in each recursive call we have $|A - M| \leq |A|/2$. This will yield a total time of $\log n[(ntd) + T(n+1, O(d), \varepsilon)]$. Since $t = c\log(n)/\varepsilon$, $d = \log^2 n$ and $\varepsilon = \frac{\Theta(1)}{d}$, it follows that the time is at most $n\log^5(n) + \log(n)T(n+1, O(d), \varepsilon)$, and the theorem follows.

To show that $|A - M| \leq |A|/2$, first observe that if the algorithm reaches step (2), then for a large enough constant $c > 1$ it holds, with high probability, that the set H of hitting vectors a has cardinality at most $\varepsilon \cdot n_A$, as otherwise one such vector would have been sampled. Thus, the subroutine ALG returns a map where the vast majority of the points $f(a)$ have been matched to a point $f(b)$ such that $\|f(a) - f(b)\|_1 = 4d + 2$. More formally, the cost of the mapping g is

$$\begin{aligned} C &= \sum_{a" \in A"} \|a" - g(a")\|_1 \\ &\leq (1+\varepsilon)[n_A(4d+2) + 2|H|] \\ &\leq (1+\varepsilon)[n_A(4d+2) + 2\varepsilon n_A] \\ &\leq n_A(4d+2) + 4\varepsilon n_A(d+2) \\ &\leq n_A(4d+2) + n_A \end{aligned}$$

where in the last step we used the assumption about ε.

Denote $m = |M|$. Observe that the cost C of the mapping g can be alternatively written as:

$$C = m(4d+2) + (n_A - m)(4d+4) = n_A(4d+4) - 2m$$

This implies $m = (n_A(4d+4) - C)/2$. Since we showed earlier that $C \leq n_A(4d+2) + n_A$, we conclude that

$$m = (n_A(4d+4) - C)/2 \geq (2n_A - n_A)/2 = n_A/2.$$

Thus, $|A - M| = n_A - m \leq n_A/2$, completing the proof.

□

3.4 Experiments

We perform an empirical evaluation of our Chamfer distance estimation algorithm.

Summary of Results Our experiments demonstrate the effectiveness of our algorithm for both low and high dimensional datasets and across different dataset sizes. Overall, it is much faster than brute force (even accelerated with KD-trees). Further, our algorithm is both faster and more sample-efficient than uniform sampling. It is also robust to different datasets: while uniform sampling performs well for most datasets in our experiments, it performs poorly on datasets where the distances from points in A to their neighbors in B vary significantly. In such cases, our algorithm is able to adapt its importance sampling probabilities appropriately and obtain significant improvements over uniform sampling.

Dataset	$\|A\|, \|B\|$	d	**Experiment**	**Metric**	**Reference**
ShapeNet	$\sim 8 \cdot 10^3, \sim 8 \cdot 10^3$	3	Small Scale	ℓ_1	[53]
Text Embeddings	$2.5 \cdot 10^3, 1.8 \cdot 10^3$	300	Small Scale	ℓ_1	[127]
Gaussian Points	$5 \cdot 10^4, 5 \cdot 10^4$	2	Outliers	ℓ_1	-
DEEP1B	$10^4, 10^9$	96	Large Scale	ℓ_2	[20]
Microsoft-Turing	$10^5, 10^9$	100	Large Scale	ℓ_2	[176]

Table 3.1: Summary of our datasets. For ShapeNet, the value of $|A|$ and $|B|$ is averaged across different point clouds in the dataset.

3.4.1 Experimental Setup

We use three different experimental setups, small scale, outlier, and large scale. They are designed to 'stress test' our algorithm, and relevant baselines, under vastly different parameter regimes. The datasets we use are summarized in Table 3.1. For all experiments, we introduce uniform sampling as a competitive baseline for estimating the Chamfer distance, as well as (accelerated) brute force computation. All results are averaged across 20+ trials and 1 standard deviation error bars are shown when relevant.

Small Scale These experiments are motivated from common use cases of Chamfer distance in the computer vision and NLP domains. In our small scale experiments, we use two different datasets: (a) the ShapeNet dataset, a collection of point clouds of objects in three dimensions [53]. ShapeNet is a common benchmark dataset frequently used in computer graphics, computer vision, robotics and Chamfer distance is a widely used measure of similarity between different ShapeNet point clouds [53]. (b) We create point clouds of words from text documents from [127]. Each point represents a word embedding obtained from the word-to-vec model of [151] in $\mathbb{R}^{300}$ applied to the Federalist Papers corpus. As mentioned earlier, a popular relaxation of the common Earth Mover Distance is exactly the (weighted) version of the Chamfer distance [17, 127].

Since ShapenNet is in three dimensions, we implement nearest neighbor queries using KD-trees to accelerate the brute force baseline as KD-trees can perform exact nearest neighbor search quickly in small dimensions. However, they have runtime exponential in dimension meaning they cannot be used for the text embedding dataset, for which we use a standard naive brute force computation. For both these datasets, we implement our algorithms using Python 3.9.7 on a M1 MacbookPro with 32GB of RAM. We also use an efficient implementation of KD trees in Python and use Numpy and Numba whenever relevant. Since the point clouds in the dataset have approximately the same n value, we compute the symmetric version $\texttt{CH}(A, B) + \texttt{CH}(B, A)$. For these experiments, we use the ℓ_1 distance function.

Outliers This experiment is meant to showcase the robustness of our algorithm. We consider two point clouds, A and B, each sampled from Gaussian points in $\mathbb{R}^{100}$ with identity covariance. Furthermore, we add an "outlier" point to A equal to $0.5n \cdot \mathbf{1}$, where $\mathbf{1}$ is the all ones vector.

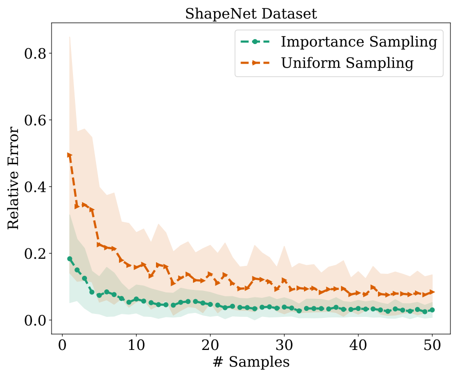
This example models scenarios where the distances from points in A to their nearest neighbors in B vary significantly, and thus uniform sampling might not accurately account for all distances, missing a small fraction of large ones.

Large Scale The purpose of these experiments is to demonstrate that our method scales to datasets with billions of points in hundreds of dimensions. We use two challenging approximate nearest neighbor search datasets: DEEP1B [20] and Microsoft Turing-ANNS [176]. For these datasets, the set A is the query data associated with the datasets. Due to the asymmetric sizes, we compute $\texttt{CH}(A, B)$. These datasets are normalized to have unit norm and we consider the ℓ_2 distance function.

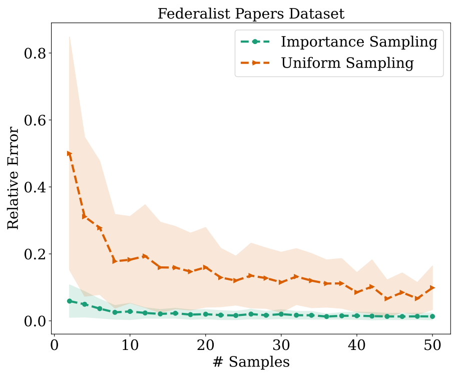
These datasets are too large to handle using the prior configurations. Thus, we use a proprietary in-memory parallel implementation of the SimHash algorithm, which is an ℓ_2 LSH family for normalized vectors according to Definition 3.2.1 [59], on a shared virtual compute cluster with 2x64 core AMD Epyc 7763 CPUs (Zen3) with 2.45Ghz - 3.5GHz clock frequency, 2TB DDR4 RAM and 256 MB L3 cache. We also utilize parallelization on the same compute cluster for naive brute force search.

3.4.2 Results

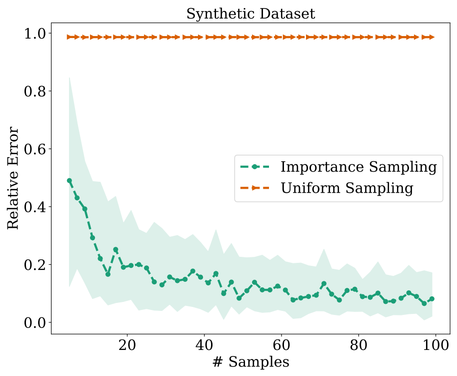
Small Scale First we discuss configuring parameters. Recall that in our theoretical results, we use $O(\log n)$ different scales of the LSH family in `CrudeNN`. `CrudeNN` then computes (over)



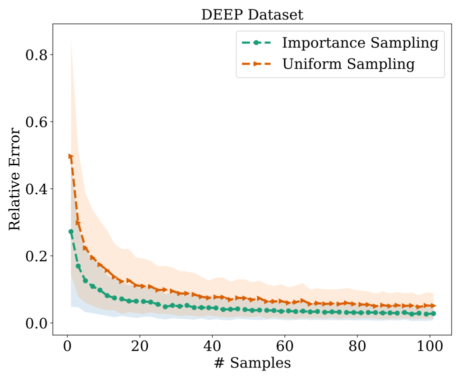
(a) ShapeNet



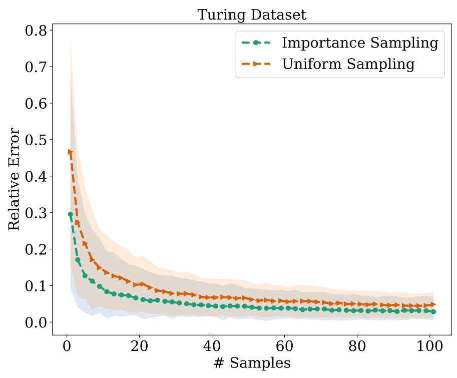
(b) Federalist Papers



(c) Gaussian Points

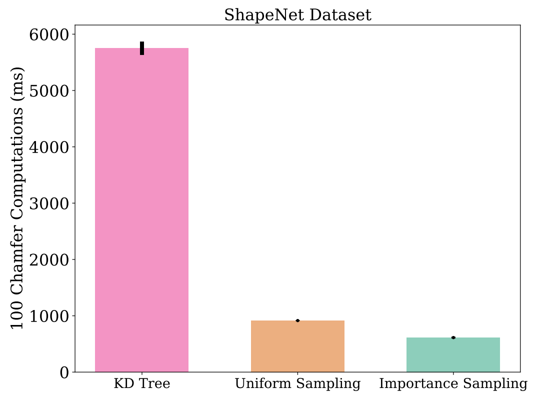


(d) DEEP

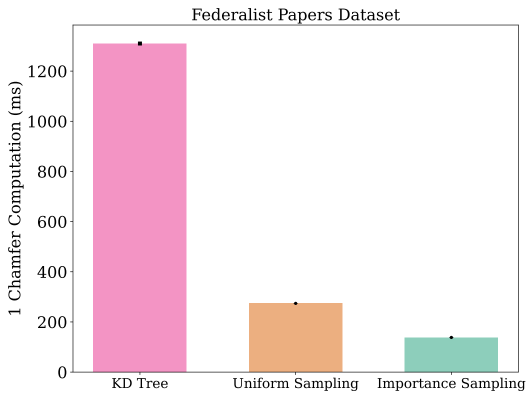


(e) Turing

Figure 3.1: Sample complexity vs relative error curves.



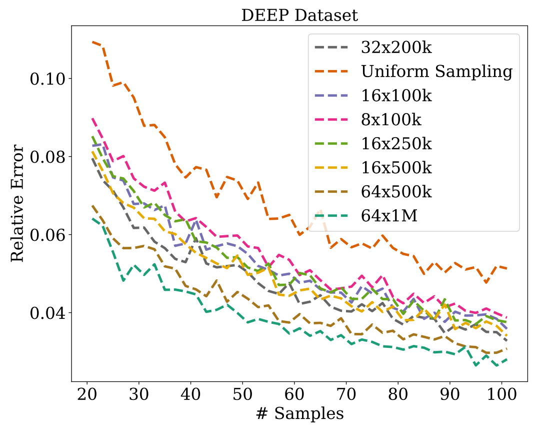
(a) ShapeNet



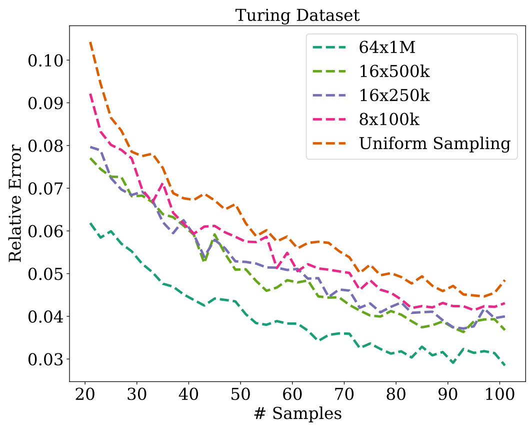
(b) Federalist Papers

Figure 3.2: Runtime experiments. We set the number of samples for uniform and importance sampling such that the relative errors of their respective approximations are similar.

estimates of the nearest neighbor distance from points in A to B (in near linear time) which is then used for importance sampling by `Chamfer-Estimate`. Concretely for the ℓ_1 case,



(a) DEEP



(b) Turing

Figure 3.3: Sample complexity vs relative error curves as we vary the number of LSH data structures and window sizes. Each curve maps $k \times W$ where k is the number of LSH data structures we use to repeatedly hash points in B and W is the window size, the number of points retrieved from B that hash closest to any given a at the smallest possible distance scales.

this the LSH family corresponds to imposing $O(\log n)$ grids with progressively smaller side lengths. In our experiments, we treat the number of levels of grids to use as a tuneable parameter in our implementation and find that a very small number suffices for high quality results in the importance sampling phase.

Indeed, Figure 3.4 (b) shows that only using 3 grid levels is sufficient for the crude estimates $\boldsymbol{D}_a$ to be within a factor of 2 away from the true nearest neighbor values for the ShapeNet dataset, averaged across different point clouds in the dataset. Thus for the rest of the Small Scale experiments, we fix the number of grid levels to be 3.

We now discuss the main experimental results. Figure 3.1 (a) shows the sample complexity vs accuracy trade offs of our algorithm, which uses importance sampling, compared to uniform sampling. Accuracy is measured by the relative error to the true value. We see that our algorithm possesses a better trade off as we obtain the same relative error using only 10 samples as uniform sampling does using 50+ samples, resulting in at least a **5x** improvement in sample complexity. For the text embedding dataset, the performance gap between our importance sampling algorithm and uniform sampling grows even wider, as demonstrated by Figure 3.1 (b), leading to $>$ **10x** improvement in sample complexity.

In terms of runtimes, we expect the brute force search to be much slower than either importance sampling and uniform sampling. Furthermore, our algorithm has the overhead of first estimating the values $\boldsymbol{D}_a$ for $a \in A$ using an LSH family, which uniform sampling does not. However, this is compensated by the fact that our algorithm requires much fewer samples to get accurate estimates.

Indeed, Figure 3.2 (a) shows the average time of 100 Chamfer distance computations between randomly chosen pairs of point clouds in the ShapeNet dataset. We set the number of samples for uniform sampling and importance sampling (our algorithm) such that they both output estimates with (close to) 2% relative error. Note that our runtime includes the time to build our LSH data structures. This means we used 100 samples for importance sampling and 500 for uniform. The brute force KD Tree algorithm (which reports exact answers) is approximately 5x slower than our algorithm. At the same time, our algorithm is 50% faster than uniform sampling. For the Federalist Papers dataset (Figure 3.2 (b)), our algorithm only required 20 samples to get a 2% relative error approximation, whereas uniform sampling required at least 450 samples. As a result, our algorithm achieved **2x** speedup compared to uniform sampling.

Outliers We performed similar experiments as above. Figure 3.1 (c) shows the sample complexity vs accuracy trade off curves of our algorithm and uniform sampling. Uniform sampling has a very large error compared to our algorithm, as expected. While the relative error of our algorithm decreases smoothly as the sample size grows, uniform sampling has the same high relative error. In fact, the relative error will stay high until the outlier is sampled, which typically requires $\Omega(n)$ samples.

Large Scale We consider two modifications to our algorithm to optimize the performance of `CrudeNN` on the two challenging datasets that we are using; namely, note that both datasets are standard for benchmarking billion-scale nearest neighbor search. First, in the `CrudeNN` algorithm, when computing $\boldsymbol{D}_a$ for $a \in A$, we search through the hash buckets $h_1(a), h_2(a), \ldots$ containing a in increasing order of i (i.e., smallest scale first), and retrieve the first W (window size) distinct points in B from these buckets. Then, the whole process is repeated k times, with k independent LSH data structures, and $\boldsymbol{D}_a$ is set to be the distance from a to the closest among all Wk retrieved points.

Note that previously, for our smaller datasets, we set $\boldsymbol{D}_a$ to be the distance to the first point in B colliding with a, and repeated the LSH data structure once, corresponding to $W = k = 1$. In our figures, we refer to these parameter choices as $k \times W$ and test our algorithm across several choices.

For the DEEP and Turing datasets, Figures 3.1 (d) and 3.1 (e) show the sample complexity vs relative error trade-offs for the best parameter choice (both 64×10^6) compared to uniform sampling. Qualitatively, we observe the same behavior as before: importance sampling requires fewer samples to obtain the same accuracy as uniform sampling. Regarding the other parameter choices, we see that, as expected, if we decrease k (the number of LSH data structures), or if we decrease W (the window size), the quality of the approximations $\{\boldsymbol{D}_a\}_{a\in A}$ decreases and importance sampling has worse sample complexity trade-offs. Nevertheless, for all parameter choices, we see that we obtain superior sample complexity trade-offs compared to uniform sampling, as shown in Figure 3.3. A difference between these parameter choices are the runtimes required to construct the approximations $\{\boldsymbol{D}_a\}_{a\in A}$. For example for the DEEP dataset, the naive brute force approach (which is also optimized using parallelization) took approximately $1.3 \cdot 10^4$ seconds, whereas the most expensive parameter choice of 64×10^6 took approximately half the time at 6.4×10^3 and the cheapest parameter choice of 8×10^5

took 225 seconds, leading to a **2x-50x** factor speedup. The runtime differences between brute force and our algorithm were qualitative similar for the Turing dataset.

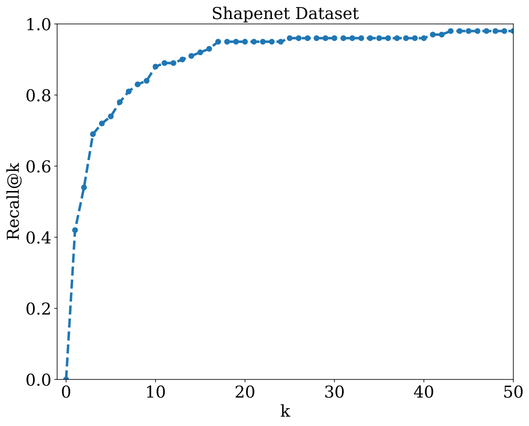
Similar to the small scale dataset, our method also outperforms uniform sampling in terms of runtime if we require they both output high quality approximations. If we measure the runtime to get a 1% relative error, the $16 \times 2 \cdot 10^5$ version of our algorithm for the DEEP dataset requires approximately 980 samples with total runtime approximately 1785 seconds, whereas uniform sampling requires > 1750 samples and runtime > 2200 seconds, which is $> 23\%$ slower. The gap in runtime increases if we desire approximations with even smaller relative error, as the overhead of obtaining the approximations $\{\boldsymbol{D}_a\}_{a \in A}$ becomes increasingly overwhelmed by the time needed to compute the exact answer for our samples.

Additional Experimental Results For the ShapeNet dataset, we show we can efficiently recover the true exact nearest neighbor of a fixed point cloud A in Chamfer distance among a large collect of different point clouds. In other words, it is beneficial for finding the 'nearest neighboring point cloud'. Recall the ShapeNet dataset, contains approximately $5 \cdot 10^4$ different point clouds. We consider the following simple (and standard) two step pipeline: (1) use our algorithm to compute an approximation of the Chamfer distance from A to every other point cloud B in our dataset. More specifically, compute an approximation to $\texttt{CH}(A, B) + \texttt{CH}(B, A)$ for all B using 50 samples and the same parameter configurations as the small scale experiments. Then filter the dataset of points clouds and prune down to the top k closest point cloud candidates according to our approximate distances. (2) Find the closest point cloud in the top k candidates via exact computation.

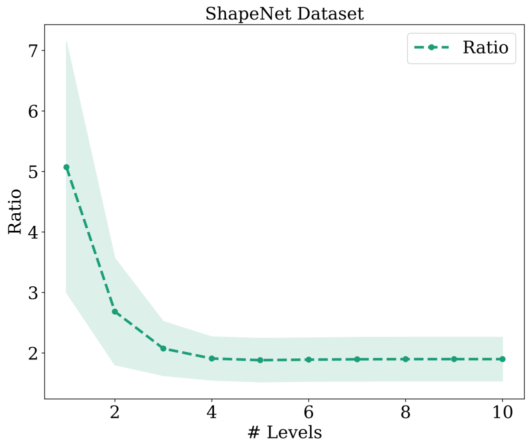
We measure the accuracy of this via the standard recall @k measure, which computes the fraction of times the *exact* nearest neighbor B of A, averaged over multiple A's, is within the top k choices. Figure 3.4 (a) shows that the true exact nearest neighbor of A, that is the point cloud B which minimizes $\texttt{CH}(A, B) + \texttt{CH}(B, A)$ among our collection of multiple point clouds, is within the top 30 candidates $> 98\%$, time (averaged over multiple different choices of A). This represents a more than **1000x** reduction in the number of point clouds we do exact computation over compared to the naive brute force method, demonstrating the utility of our algorithm for downstream tasks.

3.5 Conclusion

We present an efficient approximation algorithm for estimating the Chamfer distance up to a $1 + \varepsilon$ factor in time $O\left(nd \log(n)/\varepsilon^2\right)$. The result is complemented with a conditional lower bound which shows that reporting a Chamfer distance mapping of similar quality requires nearly quadratic time. Our algorithm is easy to implement in practice and compares favorably to brute force computation and uniform sampling. We envision our main tools of obtaining fast estimates of coarse nearest neighbor distances combined with importance sampling can have additional applications in the analysis of high-dimensional, large scale data.



(a) ShapeNet NNS pipeline experiments



(b) Quality of approximations $\boldsymbol{D}_a$ vs the number of levels of LSH data structure

Figure 3.4: Additional figures for the ShapeNet dataset.

Chapter 4

Dimensionality Reduction for Sparse Vectors

In this chapter, we prove new dimensionality reduction results for preserving pairwise distances of sparse vectors. We recall the three guiding questions of this chapter introduced in Section 1.3:

> *(Q1) Can we obtain improved dimensional reduction for preserving pairwise distances between sparse vectors for general ℓ_p norms?*
>
> *(Q2) Can we obtain improved dimensionality reduction for non-negative sparse vectors in general ℓ_p norms (such as ℓ_∞)?*
>
> *(Q3) What are the limitations of linear maps in dimensionality reduction for sparse vectors?*

In addition, we introduce another question we study in this chapter. The lower bounds used to answer Q3 above are shown by demonstrating specific sparse datasets where preserving *all pairs* simultaneously is provably impossible. Therefore, it is natural to ask if relaxed guarantees can circumvent the strong lower bounds we show for Question (3). Towards this, we if we only require the distances between constantly many pairs to be preserved (for example, 99% of the pairwise distance)? Can this lead to better upper bounds? In this 'average-case' setting where we only care about preserving the distance between a large constant fraction of pairs, there exists a positive answer based on a folklore dimensionality reduction map.

Definition 4.0.1 (Birthday Paradox Map). *Consider the linear map $f : \mathbb{R}^d \to \mathbb{R}^m$ where every coordinate in $\{1, \ldots, d\} = [d]$ is mapped uniformly at random to one of m coordinates. The coordinates in $[d]$ that are mapped to the same 'bucket' in $[m]$ are summed.*

For any fixed s-sparse vector $x \in \mathbb{R}^d$, the birthday paradox implies that if $m \geq Cs^2$ for a large constant C, then with probability at least 99%, all the support elements of x do not collide. Thus with probability at least 99%, $\|f(x)\|_p = \|x\|_p$ holds for any fixed x. Clearly, the upper bound does not depend on p and works for general sparse vectors. The natural question following from this discussion is if we can improve upon the $O(s^2)$ upper bound of the birthday paradox map. This is the last question we address.

> *(Q4) Is the birthday paradox max optimal for 'average-case' dimensionality reduction of sparse vectors?*

4.1 Our Contributions

We detail our contributions for each question below. To summarize, Table 4.1 lists our upper bound results towards Questions (1) and (2), Table 4.2 lists our lower bound results for Questions (3) and (4). Finally, Table 4.3 lists some applications of our upper bounds.

4.1.1 Question 1

We recall our first result. It is an embedding for general sparse vectors.

Theorem 1.3.1. (Informal, see Theorem 4.4.2) *For every $p \geq 1$, there exists a linear map $f : \mathbb{R}^d \to \mathbb{R}^m$ for $m = s^{p+2}\log(s)\log(d/\varepsilon)2^{O(p)}/\varepsilon^2$ which satisfies $\|f(x)\|_p = (1 \pm \varepsilon)\|x\|_p$ for all s-sparse vectors $x \in \mathbb{R}^d$.*

In comparison, the best prior bound achieves embedding dimension $p^{O(p)}s^p\log^{p-1}(d)/\varepsilon^2$, also via a linear map. While the exponent of s is smaller by 2 in the prior bound, it incurs an exponential dependence on p in both the terms $p^{O(p)}$ and $\log(d)^{p-1}$. Thus, qualitatively, we obtain improvements when the sparsity is small (for example $s \lesssim \log^{p-3}(d)$). Quantitatively, our improvement can be seen when we embed ℓ_∞ into a large ℓ_p norm for sparse vectors. The choice of $p = O(\log(s)/\varepsilon)$ is the natural choice as the $\ell_{O(\log(s)/\varepsilon)}$ norm approximates the ℓ_∞ norm up to a multiplicative $1 \pm \varepsilon$ factor for s-sparse vectors. We have the following corollary.

Corollary 4.1.1 (Informal, see Corollary 4.5.5)**.** *Let $p = O(\log(s)/\varepsilon)$. There is a linear map $f : \mathbb{R}^d \to \mathbb{R}^m$ such that $\|f(x)\|_p = (1 \pm \varepsilon)\|x\|_\infty$ for all s-sparse $x \in \mathbb{R}^d$ for the following m:*

- *If we use the embedding of [209], then $m = s^{O(\varepsilon^{-1}(\log\log d + \log s))}/\varepsilon^2$.*
- *If we use the embedding of Theorem 4.4.2, then $m = \log(s)\log(d/\varepsilon)s^{O(\varepsilon^{-1}\log s)}/\varepsilon^2$.*

Thus, our embedding avoids a multiplicative overhead of $s^{O(\varepsilon^{-1}\log\log d)}$.

4.1.2 Question 2

Perhaps the most conceptually interesting contribution of this chapter is the following theorem for embedding non-negative sparse vectors.

Theorem 1.3.2. (Informal, see Theorem 4.5.2) *Let X be a set of n non-negative s-sparse vectors. For every $p \geq 1$, there exists a **non-linear** map $f : \mathbb{R}^d \to \mathbb{R}^m$ for $m = O(\log(n) \cdot \min(s^2/\varepsilon^2, s/\varepsilon^3))$ which satisfies $\|f(x) - f(y)\|_p = (1 \pm \varepsilon)\|x - y\|_p$ for all $x, y \in X$. For the ℓ_∞ case, we can instead guarantee $\|f(x) - f(y)\|_\infty = \|x - y\|_\infty$ for all $x, y \in X$ with $m = O(s \log(n))$.*

There are three interesting aspects of this result that we want to highlight. They establish important context for the lower bound results introduced shortly: (a) the map f is non-linear, (b) X needs to be a point sets of non-negative sparse vectors, and (c), our embedding is valid for ℓ_∞.

Before we discuss lower bounds, we present some interesting consequences of Theorem 1.3.2. First, similar ideas extend to embeddings for *general* sparse vectors, but with bounded entries.

Theorem 4.1.2 (Informal, see Theorem 4.5.4)**.** *Let X be a set of n s-sparse vectors with entries in $\{-\Delta, \ldots, 0, \ldots, \Delta\}$. For every $p \geq 1$, there exists a non-linear map $f : \mathbb{R}^d \to \mathbb{R}^m$ for $m = O(s^2 \Delta^{O(p)} \log(n)/\varepsilon^2)$ which satisfies $\|f(x) - f(y)\|_p = (1 \pm \varepsilon)\|x - y\|_p$ for all $x, y \in X$.*

Quantitatively, the above result allows us to embed ℓ_∞ into a much smaller ℓ_p dimension for $p = O(\log(s)/\varepsilon)$ for *general* sparse vectors, assuming the entries are bounded.

Corollary 4.1.3 (Informal, see Corollary 4.5.5)**.** *Consider the setting of Corollary 4.1.1 with a dataset X of s-sparse vectors. If the entries of X are in $\{-O(1), \ldots, O(1)\}$, then there is a map with $m = \log(|X|) s^{O(1/\varepsilon)}/\varepsilon^2$.*

The corollary suggests that the 'hardness' for embedding general sparse vectors maybe due to entries with a large range.

4.1.3 Question 3

We show that our upper bound for embedding non-negative sparse vectors presented in Theorem 1.3.2 is optimal in many ways.

Non-linearity is required. We give an example of a $O(1)$-sparse point set where any *linear map* requires $\Omega(d)$ dimensions to preserve the ℓ_∞ norm up to relative error 0.5.

Theorem 4.1.4. *Let S be the set of all 10-sparse vectors in $\mathbb{R}^d$ with all non-zero coordinates being equal to 1. Let $A : \mathbb{R}^d \to \mathbb{R}^m$ be a matrix such that*

$$\frac{1}{2}\|x\|_\infty \leq \|Ax\|_\infty \leq \frac{3}{2}\|x\|_\infty \quad \forall x \in S.$$

Then $m = \Omega(d)$.

Point Set	**Norm**	**Our Embedding Dimension**	**Thm.**	**Prior Embedding Dimension**
All s-sparse vectors	Fixed ℓ_p $p \geq 1$	$\left(2^{O(p)} s^{p+2} \log(s) \log(d/\varepsilon)\right) \cdot \frac{1}{\varepsilon^2}$ (linear map)	4.4.2	$\left(p^{O(p)} s^p \log^{p-1}(d)\right) \cdot \frac{1}{\varepsilon^2}$ (linear map, $p > 1, p \neq 2$) [209]
Set of n non-negative s-sparse vectors	All ℓ_p	$O\left(\frac{\log(n)}{\varepsilon} \cdot \min\left(\frac{s^2}{\varepsilon}, \frac{s}{\varepsilon^2}\right)\right)$ (non-linear map)	4.5.2	$O\left(\frac{s^2(p\log(n))^{p-1}}{\varepsilon^2}\right)$ (linear map and fixed p) [209]
Set of n non-negative s-sparse vectors	ℓ_∞	$O\left(s \log(n)\right)$ (non-linear map)	4.5.2	–

Table 4.1: Results are stated for preserving all pairwise distances up to a multiplicative $(1 \pm \varepsilon)$-factor for vectors in the point set. The last row is an exact embedding. The result of n non-negative vectors listed in the second row is not explicitly given in [209], but can be easily derived from their argument. The $p = 1, 2$ cases were originally addressed in [35] and [44] respectively and $O(\varepsilon^{-2} s \log(d/s))$ dimensions suffice for these two cases. We note that our Theorem 4.5.2 satisfies the additional guarantee that it also preserves the norm of the pairwise *sums* in ℓ_∞ up to a factor of 2.

'Non-smoothness' is required. The map f of Theorem 1.3.2 is continuous but not differentiable. Motivated by this phenomenon (recall that most dimensionality reduction maps in literature are linear), we show that any upper bound satisfying Theorem 1.3.2 cannot be 'very smooth': any twice differentiable map (satisfying the same hypothesis) must embed into $\Omega(d)$ dimensions.

Theorem 4.1.5. *Suppose $f : \mathbb{R}^d \to \mathbb{R}^m$ is such that $f(x) = (f_1(x), \ldots, f_m(x))$ where each f_i is twice differentiable with continuous second partial derivatives. Let S be the set of all 10-sparse vectors in $\mathbb{R}^d$ with all non-zero coordinates being equal to 1. Suppose f satisfies*

$$0.99\|rx\|_\infty \leq \|f(rx)\|_\infty \leq 1.01\|rx\|_\infty \quad \forall r > 0, \forall x \in S.$$

Then $m = \Omega(d)$.

Sums cannot be preserved. It is natural to ask if the map f of Theorem 1.3.2 can also preserve *sums* of non-negative sparse vectors: $\|f(x)+f(y)\|_p \approx \|x+y\|_p$?. Indeed, our upper bound of Theorem 1.3.2 has the additional property that it preserves the norms of sums in ℓ_∞ norm up to a multiplicative factor of 2, even if we embed into *one* dimension. We show that the approximation factor 2 is tight in a very strong sense.

Theorem 4.1.6. *Let e_i be the ith basis vector in $\mathbb{R}^d$. Consider the set $S = \{e_i\} \cup \{0\}$ of $d+1$ vectors. Suppose $f : S \to \mathbb{R}^m$ be an arbitrary mapping which satisfies*

$$\|x + y\|_\infty \leq \|f(x) + f(y)\|_\infty \leq (2 - \varepsilon)\|x - y\|_\infty \quad \forall x, y \in S$$

for any $\varepsilon > 0$. Then $m \geq d$.

General sparse vectors are hard to embed. Lastly, we consider dropping the non-negative hypothesis. We again show that for the ℓ_∞ case, upper bounds as in the non-negative case cannot exist for general sparse vectors. Note that prior lower bounds studied in [209] are restricted to linear maps. In contrast, we show in the following theorem that any map which satisfies the same hypothesis as Theorem 1.3.2, but which holds for general sparse vectors with possibly *negative* entries, must embed into $\Omega(d)$ dimensions.

Theorem 4.1.7. *Let e_i be the ith basis vector in $\mathbb{R}^d$. Consider the set $S = \{\pm e_i\} \cup \{0\}$ of $2d+1$ vectors. Suppose $f : S \to \mathbb{R}^m$ be an arbitrary mapping which satisfies*

$$0.9\|x - y\|_\infty \leq \|f(x) - f(y)\|_\infty \leq 1.1\|x - y\|_\infty \quad \forall x, y \in S,$$

$$\|x + y\|_\infty \leq \|f(x) + f(y)\|_\infty \leq C\|x + y\|_\infty \quad \forall x, y \in S$$

for any $C \geq 1$. Then $m \geq d$.

Lastly, we can ask for the right dependence on s and ε for Theorem 4.5.2. The following lower bound implies that $\Omega(s/\log(n))$ is necessary, matching the s-dependency of our upper bound.

Theorem 4.1.8 (Informal, see Theorem 4.6.1)**.** *Any map f satisfying similar guarantees to that of Theorem 4.5.2 with $\varepsilon = O(1)$ must map to $\Omega(s/\log(n))$ dimensions.*

4.1.4 Question 4

Our main contribution towards Question (4) is to demonstrate that the average-case guarantees of the folklore birthday paradox upper bound of Definition 4.0.1 is *optimal* in many natural settings. First recall that the folklore upper bound guarantees that for any p and for any point set of s-sparse vectors, there exists an embedding into $O(s^2)$ dimensions which preserves the ℓ_p norm *exactly* for 99% of the vectors (this is a weaker hypothesis than requiring pairwise distances to preserved as long as all zeros vector, which is s-sparse, is also mapped to the all zeros vector).

Arbitrary linear maps. Our first result states that any linear map with arbitrary real entries which satisfies the properties of the folklore upper bound must embed to at least $m = \Omega(s^2)$ dimensions. Our lower bound holds for any even integer p. More precisely, we construct a set of s-sparse vectors in d-dimensions, where we show that any linear map A which guarantees exact norm-preservation for more than 99% of this set (i.e. a randomly chosen vector in the set has a $> 99\%$ chance of their ℓ_p norm being preserved under the map) must map to $\Omega(s^2)$ dimensions. We note that proving lower bounds for linear maps is motivated by the fact that in settings such as streaming algorithms, *any* algorithm may as well be a linear mapping (under some restrictions) [140].

Theorem 4.1.9 (Informal, see Theorem 4.7.1)**.** *Let $p \geq 2$ be an even integer. There exists a point set $S \subset \mathbb{R}^{s^2}$ of s-sparse vectors such that any linear map $A : \mathbb{R}^{s^2} \to \mathbb{R}^m$ satisfying $\|Ax\|_p = \|x\|_p$ for 99% of vectors $x \in S$ must map to $m = \Omega(s^2)$ dimensions.*

Beyond Exact Preservation. Our lower bound for linear maps extends to a stronger statement for the ℓ_2 case. We show that $m = \Omega(s^2)$ even if we only require the *weaker* guarantee of $|\|Ax\|_2^2 - \|x\|_2^2| \lesssim \frac{\|x\|_2^2}{s}$. This is optimal as any weaker relative error of $\varepsilon = \omega(1/s)$ can be accomplished with JL with $\tilde{O}(1/\varepsilon^2) = o(s^2)$ dimensions. Note the folklore upper bound guarantees the stronger statement of *exact* preservation; however for lower bounds, assuming a weaker hypothesis implies a stronger result.

Theorem 4.1.10 (Informal, see Theorem 4.2.1)**.** *There exists a point set $S \subset \mathbb{R}^{s^2}$ of s-sparse vectors such that any linear map $A : \mathbb{R}^{s^2} \to \mathbb{R}^m$ satisfying $|\|Ax\|_2^2 - \|x\|_2^2| \leq O(\|x\|_2^2/s)$ for 99% of vectors $x \in S$ must map to $m = \Omega(s^2)$ dimensions.*

We also extend this lower bound for linear maps to the case where we require inner-products to be preserved, rather than ℓ_2 norms (Theorem 4.7.4).

Arbitrary smooth mappings. For the ℓ_2 case, we consider a more general class of mappings $f : \mathbb{R}^{s^2} \to \mathbb{R}^m$ where $f(x) = (f_1(x), \ldots, f_m(x))$ and each $f_i : \mathbb{R}^{s^2} \to \mathbb{R}$ is twice differentiable with continuous second partial derivatives. In this case, we can prove the following theorem. We prove that this large richer class of mappings still requires $m = \Omega(s^2)$ to satisfy the guarantees of the folklore upper bound for the ℓ_2 case.

Theorem 4.1.11 (Informal, see Theorem 4.7.2)**.** *The lower bound statement of Theorem 4.1.10 extends to the general class of mappings defined above.*

Beyond Embeddings. Our ℓ_2 lower bounds hold even when we are not restricted to computing the norm on the embeddings produced by a mapping f and can use another function g (on top of the output of f) to compute the norm.

Formally, we define an encoding-decoding scheme using an *encoding* function f which maps a s-sparse vector in $\mathbb{R}^{s^2}$ to a dimension m. A *decoding* function g then takes the output of f and maps it to a potentially much larger dimension c. Our goal is to show that even if $c \gg s^2$, as long as m is substantially less than the initial sparsity squared, we cannot have $\|g(f(x))\|_2 \approx \|x\|_2$.

Definition 4.1.1. *We suppose the encoder and decoder functions satisfy the following.*

- *(Encoder function)* $f : \mathbb{R}^{s^2} \to \mathbb{R}^m$ *where*

$$f(x) = (f_1(x), \ldots, f_m(x))$$

  and each $f_i : \mathbb{R}^{s^2} \to \mathbb{R}$ *is twice differentiable with continuous second partial derivatives.*

- *(Decoder function)* $g : \mathbb{R}^m \to \mathbb{R}^{s^2}$ *where*

$$g(x) = (g_1(x), \ldots, g_{s^2}(x))$$

  and each $g_i : \mathbb{R}^m \to \mathbb{R}$ *is twice differentiable with continuous second partial derivatives.*

Theorem 4.1.12 (Informal, see Theorem 4.7.3)**.** *The lower bound statement of Theorem 4.1.10 extends to the case of encoder/decoder schemes of Definition 4.1.1 and shows that* f *(the encoder) must map to* $\Omega(s^2)$ *dimensions.*

Perhaps surprisingly, the theorem above states that if f and g are both sufficiently smooth, then f still must map to $\Omega(s^2)$ dimensions. That is, the whole procedure is still 'bottle-necked' by the dimension of f and this novel lower bound implies that our folklore upper bound is in fact optimal.

One reason for why this lower bound is surprising is that the hypothesis that g is smooth is quite crucial. In fact, the lower bound cannot hold if g is arbitrary, even if f is constrained to be a linear map. Indeed, compressed sensing algorithms tells us that there exists a suitable linear map $A : \mathbb{R}^d \to \mathbb{R}^{\tilde{O}(s)}$ and a decoding function g such that $g(Ax) = x$ for any s-sparse x. However, g is usually a complicated optimization step. For example in the popular Lasso algorithm for exact recovery, g amounts to solving a linear program [83], and the optimal solutions to such optimization programs can be highly discontinuous.

Specifically, Theorem 4.7.3 reveals an interesting separation result in the context of compressed sensing: if we let the decoding algorithm be arbitrary, then we can map to $\tilde{O}(s)$ dimensions. On the other hand, imposing even mild smoothness assumptions on g implies a much larger $\Omega(s^2)$ lower bound.

Point Set	**Gurantee**	**Our Lower Bound**	**Thm.**
$O(1)$-sparse non-negative vectors	$\frac{\|x\|_\infty}{2} \le \|f(x)\|_\infty \le \frac{3\|x\|_\infty}{2}$ Linear f	$\Omega(d)$	4.1.4
$O(1)$-sparse non-negative vectors	$\|x+y\|_\infty \le \|f(x)+f(y)\|_\infty \le (2-\varepsilon)\cdot\|x+y\|_\infty$ Arbitrary f, any $\varepsilon > 0$	$\Omega(d)$	4.1.6
$O(1)$-sparse non-negative vectors	$\forall r > 0, 0.99\|rx\|_\infty \le \|f(rx)\|_\infty \le 1.01\|rx\|_\infty$ Arbitrary f with continuous second-order derivatives	$\Omega(d)$	4.1.5
$O(1)$-sparse vectors	$0.9\cdot\|x-y\|_\infty \le \|f(x)-f(y)\|_\infty \le 1.1\cdot\|x-y\|_\infty$ $\|x+y\|_\infty \le \|f(x)+f(y)\|_\infty \le C\cdot\|x+y\|_\infty$ Arbitrary f, any $C > 0$	$\Omega(d)$	4.1.7
s-sparse vectors	$\|f(x)-f(y)\|_\infty \le \|x-y\|_\infty$ $C\|x-y\|_1 \le \|f(x)-f(y)\|_1$ Arbitrary f, any $C > 0$	$\Omega(Cs)$	4.6.1
s-sparse vectors	$\|f(x)\|_p = \|x\|_p$ for 99% of the points Linear f, even p	$\Omega(s^2)$	4.7.1
s-sparse vectors	$\big|\|f(x)\|_2^2 - \|x\|_2^2\big| \le O(\|x\|_2^2/s)$ for 99% of the points Arbitrary f with continuous second-order derivatives	$\Omega(s^2)$	4.7.2
s-sparse vectors	$\big|\|g(f(x))\|_2^2 - \|x\|_2^2\big| \le O(\|x\|_2^2/s)$ for 99% of the points Arbitrary f, g with continuous second-order derivatives	$\Omega(s^2)$	4.7.3

Table 4.2: Our lower bound results. The first four guarantees hold for every pair of vectors x, y in the point set specified in the respective theorems. The last column is the dimension that f must map to. In the last row, g can map to an arbitrary dimension $\gg s^2$.

4.1.5 Applications of Our Embeddings

Our dimensionality reduction results, just as the JL lemma, have a slew of downstream algorithmic applications. We present a (possibly very small) subset of some of the new applications here, with a focus on more fundamental geometric optimization problems that have been well studied[1].

[1]the selection of applications presented is heavily biased by our own interests.

For the rest of the section, we assume our input is a dataset X of n non-negative s-sparse vectors in $\mathbb{R}^d$. Many of the statements below are also applicable to general sparse vectors (with appropriate modifications), but we focus on non-negative sparse vectors for simplicity.

At a high level, dimensionality reduction allows us to make black-box use of any existing algorithm for a geometric task in low-dimensions. If the dimensionality reduction step is sufficiently powerful, we can hope to get faster runtimes at the cost of a small approximation factor loss. A prototypical example is computing the diameter of a point set, defined as follows:

Definition 4.1.1. $\text{diameter}_p = \max_{x,y \in X} \|x - y\|_p$.

Invoking our dimensionality reduction with existing algorithms for computing the diameter implies the following.

Theorem 4.1.13. *Let X be dataset of non-negative s-sparse vectors. We have the following algorithms:*

1. *Using [52], for any constant integer p, we can compute a $1+\varepsilon$ approximation to diameter_p of X in time $\tilde{O}(n/\sqrt{\varepsilon} + 2^{O(s^2 \log(1/\varepsilon))})$ which is correct with probability 99%.*
2. *We can exactly compute the diameter of X in ℓ_∞ norm in time $O(ns)$. This algorithm can be implemented in a stream using $O(s)$ words of memory.*
3. *We can exactly compute the diameter of X in ℓ_1 norm in time $n2^{O(s)}$. This algorithm can be implemented in a stream using $2^{O(s^2)}$ words of memory.*

We note additional applications to other problems such as max cut and various formulations of clustering such as k-means. They are summarized in Table 4.3.

Problem	**Definition**	**Embedding Dimension**	**Distortion**	**Reference**
Diameter	4.1.1	$O(s^2)$	1	Lemma 4.8.1 and Theorem 4.1.13
Max-Cut	4.8.1	$O(s^2/\varepsilon)$	$1 \pm \varepsilon$	Theorem 4.8.3
k-median/k-center	4.8.2	$O(s^2 \log(n)/\varepsilon^2)$	$4 \pm \varepsilon$	Theorem 4.8.5
k-means	4.8.2	$O(s^2 \log(n)/\varepsilon^2)$	$16 \pm \varepsilon$	Theorem 4.8.5
Distance Estimation	4.8.3	$O(s/\varepsilon)$	$1 \pm \varepsilon$	Theorem 4.8.6

Table 4.3: Applications of our dimensionality reduction upper bounds. In all cases, the input is a dataset X of n non-negative vectors which are s-sparse and the norm is ℓ_p for an arbitrary $p \geq 1$. The distortion bounds hold with probability at least 99%. See the theorem statements for full details.

4.1.6 Open Problems

We note some interesting questions that are not addressed by our work in this chapter.

1. What is the right ϵ and s dependency for the upper bound of Theorem 4.5.2? Can we improve the $\Omega(s/\log(n))$ lower bound of Theorem 4.6.1? For example, for every $p \geq 1$, can we construct a set of n non-negative s-sparse vectors such that any embedding f to m dimensions which preserves all pairwise ℓ_p distances up to say 1 ± 0.1 requires $m = \Omega(s/\varepsilon^2)$? For the $p = 2$ case, the JL lower bound of [11] implies a $\Omega(\log(n)/(\varepsilon^2 \log(1/\varepsilon))$ lower bound for arbitrary mappings (for a $1 \pm \varepsilon$ approximation) since their hard point set consists of basis vectors (i.e. 1-sparse). Our 'average case' lower bounds discussed in Section 4.2.2 are not directly applicable since the lower bound instances used in the proof technically require an infinite sized point set and do not rule out arbitrary mappings.

2. Can we obtain $o(s^p)$ embedding dimension for preserving the norm general s-sparse vectors? Both our upper bound of Theorem 4.4.2 and the result of [209] suffer a $s^{\Theta(p)}$ dependence in the embedding dimension (see Table 4.1). Our lower bound in Theorem 4.1.7 suggests that an exponential dependence on p is likely necessary, since we rule out the existence of any non-trivial mapping for the ℓ_∞ case. Note that [209] provide a $\Omega(s^p)$ lower bound, but only for linear maps.

3. In general, what is the power of *non*-linear embeddings in provable dimensionality reduction? As far as we are aware, all 'JL style' upper bounds use linear map. Our results demonstrate a separation between linear and non-linear maps in the natural case of non-negative sparse vectors, and it is intriguing to ask if such separations exist in other settings.

4.2 Technical Overview

For conciseness, we give on overview of our proof technique for our upper bounds for embedding non-negative sparse vectors, as well as Theorem 4.2.1 which provides lower bounds for embeddings with average case guarantees.The proofs of these results already contain many of the key ideas used in other results of the chapter.

4.2.1 Overview: Embedding Non-negative Sparse Vectors

The goal of this section is to motivate our upper bound result of Theorem 4.5.2. It states that for any dataset $X \subset \mathbb{R}^d$ of n non-negative s-sparse vectors and any $p \geq 1$, there exists a map F which embeds into $O(s^2 \log(n)/\varepsilon^2)$ dimensions and preserves all pairwise distances in ℓ_p norm. For simplicity, we only focus on the ℓ_∞ norm case in the overview. Our map F will be non-linear map.

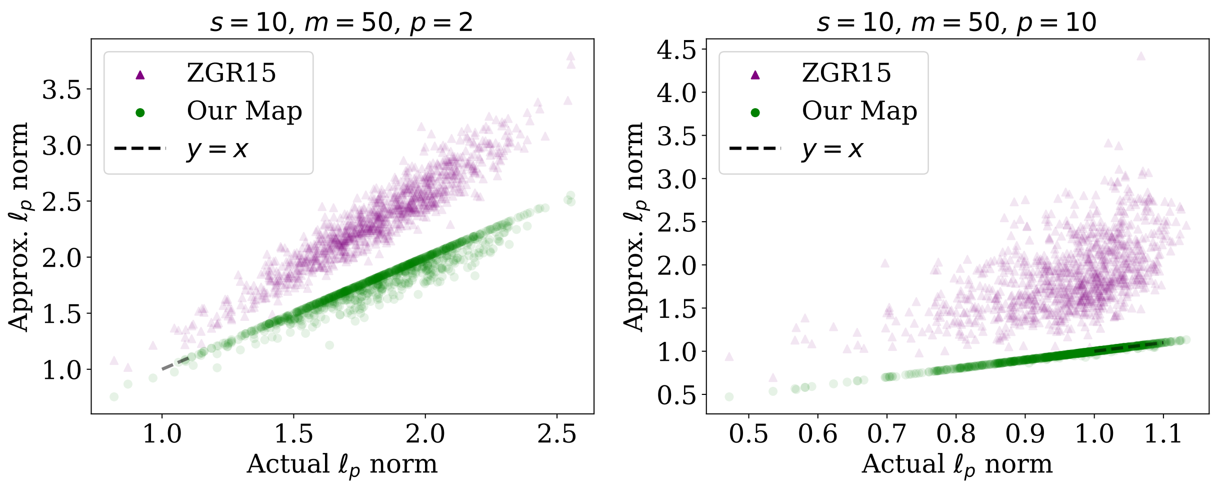


Figure 4.1: A simple plot demonstrating the performance of our non-negative embedding (Theorem 4.5.2) and the map of [209]. The dots represent a 10-sparse vectors in $\mathbb{R}^{1000}$ with entries chosen uniformly in $[0, 1]$. The x-axis is the true ℓ_p norm and the y axis is the approximated norm using the two different maps. Every vector has two dots (one for each map). We embed in $\mathbb{R}^m$ for $M = 50$ but the performance is qualitatively similar for other m. We see that the performance of our map is demonstrably superior, especially when p increases.

The main idea is to first construct a map with slightly weaker guarantees. We give a construction of a map (randomized) $f : \mathbb{R}^d \to \mathbb{R}^{O(s^2)}$ such that

1. f preserves the ℓ_∞ norm of any pair in X with probability 99%.

2. f is always never expanding: $\|f(x) - f(y)\|_\infty \leq \|x - y\|_\infty$.

Before discussing the construction of f, let's quickly see why the two points are beneficial towards the final construction. The final construction for F simply concatenates $O(\log n)$ independent copies of f. Due to independence, with high probability, every pair x, y will have at least one copy of f which 'certifies' the ℓ_∞ norm between them is at least $\|x - y\|_\infty$ (due to property (1))). Furthermore, since we know every copy of f is non-expanding, we will never overestimate the distance. Putting these two statements together implies the approximation guarantees.

Now we describe the construction for f which satisfies the two properties listed above. Property (1) actually holds for the birthday paradox map. However, this will not work work since it cannot guarantee property (2). (And more indirectly, our lower bound in Theorem 4.1.4 rules out all linear maps). Instead, we use a high non-linear map f. Similar to the birthday paradox map, we start by hashing all coordinates of the ambient dimension, d, to a set of $O(s^2)$ buckets. These buckets are the coordinates of the embedding. The crucial difference is that instead of summing the coordinates that land in a bucket, we simply take the *maximum*. More precisely, for a sparse vector x, we look at the buckets where the support elements of x land. Then for all buckets which are non-empty, we take the maximum of the support elements of x that land in the bucket. All empty buckets get 0.

The hashing step already guarantees property (1), similar to the birthday paradox upper bound. So it remains to check property (2). This reduces to checking the following: suppose coordinates $1, \dots, k$ map to the same bucket under f. Let x and y be two non-negative sparse vectors. Then we want $|\max(x_1, \dots, x_k) - \max(y_1, \dots, y_k)| \le \max_i |x_i - y_k|$, where x_i and y_i are the ith entries of x and y (they maybe 0). We can check that this is sufficient to imply property (2). To show this claim, imagine all the coordinates x_i and y_i together on the number line. They must all be to the right of the origin due to the non-negativity constraint. The right most coordinate, say x_1 without loss of generality, is closer to the rightmost coordinate among the y's, than it is to y_1. So the claim follows immediately.

Somewhat mysteriously, this construction crucially relies on the non-negativity property of the sparse vectors. An explicit example where our proposed map fails for general sparse vectors is as follows: consider two sparse vectors $x = [-1, 0, \dots, 0]$ and $y = [0, 1, 0, \dots, 0]$. Clearly, $\|x - y\|_\infty = 1$. Now suppose the first two coordinates hash to the same bucket under f. Then the first coordinate of $f(x)$ will have coordinate -1, since we take the maximum of all the support elements of x that land in the first bucket. In this case, the bucket is a singleton. Similarly, the first coordinate of the embedding of y will be 1, meaning the distance between $f(x)$ and $f(y)$ is 2. Thus, the property (2) does not hold. One could try to massage the map f a bit to fix the issue for this particular example, but our Theorem 4.1.7 rules out the existence of *any map* which preserves the distances between general sparse vectors in ℓ_∞ norm.

4.2.2 Overview: Average Case Lower Bounds

In this section, we provide insights to our lower bound result of Theorem 4.2.1. Similar ideas are used in all of our average case lower bounds so we focus on this case which we believe is the most instructive.

First we briefly outline how a known lower bounds for the JL lemma is proven. We state the proof of [11] where a slightly suboptimal lower bound is given (the result is tight up to $\log(1/\varepsilon)$ factors), but is perhaps more pedagogically useful in relation to our approach.

[11] construct a specific set of n unit vectors and consider an embedding of these points into m dimensions which preserves *all* pairwise distances up to $1 \pm \varepsilon$ multiplicative factor, or the inner product up to additive $\pm\varepsilon$. Letting $X \in \mathbb{R}^{n \times m}$ denote the matrix of embeddings, they consider the gram matrix XX^T. The key point is that they have precise control on *all* the entries of XX^T due to the JL assumption. This allows them to argue that the rank of X must be sufficiently large, implying a lower bound for m, the embedding dimension.

For us, we cannot directly employ this approach. Our lower bound hypothesis is weaker since we only guarantee that the norm is preserved say 99% of the time. This means that we have no control over a constant fraction of the entries in XX^T, which can wildly influence the rank. However, this approach suggests that *rank* is a useful parameter and this observation is the starting point of our lower bound.

We now outline our lower bound approach for the case of a linear map A given in Theorem 4.2.1. The proof for this theorem contains many of the key ideas used in our other lower bound results.

We first construct a suitable distribution over sparse vectors. Our distribution first randomly samples the support elements and then puts random Gaussian values in the sampled support. The distribution we use, $\mathrm{Unif}_{t,r}$, is defined below.

Definition 4.2.1. *Let $\mathrm{Unif}_{t,r}$ be a distribution over t-sparse vectors defined as follows. To generate $x \sim \mathrm{Unif}_{t,r}$,*

1. *First pick t coordinates uniformly at random to be the support.*

2. *The non-zero coordinates of x are i.i.d. Gaussians with variance r.*

If $r = 1$, we also denote the distribution as Unif_t. Theorem 4.2.1 states the following.

Theorem 4.2.1. *Let $A : \mathbb{R}^{s^2} \to \mathbb{R}^m$ be a linear map and $\gamma \le C/s$ for a sufficiently small constant $C > 0$. If A is such that for any $1 \le t \le s$,*

$$\Pr_{x \sim \mathrm{Unif}_t} \left(|\|Ax\|_2^2 - \|x\|_2^2| \le \gamma \|x\|_2^2 \right) \ge 0.99,$$

then $m \ge s^2/1000$.

To begin the proof, we first assume for contradiction that $m \ll s^2$. Now note that

$$\begin{aligned} \|Ax\|_2^2 - \|x\|_2^2 &= \sum_{i,j} x_i x_j \langle A_i, A_j \rangle - \sum_i x_i^2 \\ &= \mathrm{Tr}(xx^T A^T A) - \sum_i x_i^2 \end{aligned}$$

where A_i are the columns of A. For simplicity, let's ignore the $\sum_i x_i^2$ portion of P for the rest of the discussion. We also assume the sum $\sum_{i,j} x_i x_j \langle A_i, A_j \rangle$ is over $i \neq j$. (In the formal proof, we show that the $\sum_i x_i^2$ portion and the $\sum_i x_i^2 \|A_i\|_2^2$ 'roughly cancel'.) We view this expression as a polynomial $P(x)$ of degree 2 in the variables x_i, x_j. Then the condition $|\|Ax\|_2^2 - \|x\|_2^2| \le \gamma \|x\|_2^2$ implies that with large probability, $P(x)$ lies in a fixed interval I of size $O(1)$ (as typically, $\gamma \|x\|_2^2 = O(1)$). Our goal now is to show that $P(x)$ has variance $\Omega(1)$ (under the randomness of x), implying it does not lie in I with large constant probability, violating the hypothesis of the theorem.

Now it would be very bad for us if P takes on 'very small' values over a typical choice of x. This means that we cannot rule out $P \in I$. For example, P could even be equal to 0 if the non-zero entries of $x^T x$ only collide with the zero entries of $A^T A$. To avoid this possibility, we first demonstrate that a large fraction of the columns of A have norms very close to 1. However, there are s^2 columns, which are all in $\mathbb{R}^m$. Since we assumed m to be very small, this means that we have $\gg m$ almost-unit vectors in m dimensions. We show that this implies there are at least $\Omega(s^2)$ pairs of A_i and A_j that are non-zero (note that there are $\Theta(s^4)$ total pairs). This bound crucially relies on the fact that $m \ll s^2$ and can be thought of as a 'generalized' version of the rank argument used in [11]. Since any pair (i, j) is non-zero in $x^T x$ with probability approximately $1/s^2$, this means $x^T x$ and $A^T A$ 'collide' on a non-zero entry with a large constant probability.

However, this is not quite strong enough for our purposes since the coefficient of P, which are exactly $\langle A_i, A_j \rangle$, can be very close to 0 and unluckily, $x^T x$ could only collide on such small entries. Thus, our refined aim is to show that the coefficients of P that collide with the non-zero entries of $x^T x$ are 'large' in the sense that the sum of these coefficient squared is $\Omega(1)$. It can be checked that this is sufficient to show the variance of $P(x) = \Omega(1)$.

Then in the formal proof, we show that the sum of non-zero coefficient squared is $\Omega(1)$, via a 'fine-grained' exploitation of the fact that $m \ll s^2$ (i.e. another appearance of rank). Now to argue anti-concentration of $P(x)$, we invoke the following classical inequality which states that the random variable $P(x)$ cannot be concentrated in an interval that is much smaller than the variance (which is directly related to the sum of non-zero coefficients squared).

Lemma 4.2.2 (Theorem 8 in [46])**.** *Let $P : \mathbb{R}^n \to \mathbb{R}$ be a polynomial of degree d. Let Z denote the random variable where P is evaluated on a standard Gaussian vector in $\mathbb{R}^n$. There is an absolute constant B such that*

$$\Pr(|Z| \leq \varepsilon\sqrt{\mathrm{Var}(Z)}) \leq Bd\varepsilon^{1/d}.$$

An application of Lemma 4.2.2 then implies that with sufficiently large constant probability, $P(x)$ *does not* lie in the interval I defined above, contradicting the theorem hypothesis that $\|Ax\|_2^2$ approximately preserves the norm of x with large probability. Hence, our assumption that $m \ll s^2$ is not valid, finishing the proof. The formal details are given in the proof of Theorem 4.2.1.

4.3 Related Works

Lower-bounds for the JL Lemma Lower bounds for dimensionality reduction were introduced for understanding the minimum embedding dimension for n vectors with at most $1 + \varepsilon$ multiplicative distortion. By using rank arguments of perturbed identity matrices, as outlined in Section 4.2.2, the first such lower bounds showed that the embedding dimension must satisfy $\Omega(\log(n)/(\varepsilon^2 \log(1/\varepsilon)))$, even when the embedded vectors are simple basis vectors [11]. Furthermore, these bounds hold even when a non-linear or adaptive embedding function is applied; however they crucially depend on the *maximum* distortion being smaller than ε. These lower bounds were improved to an optimal $\Omega(\log(n)/\varepsilon^2)$ bound for an oblivious or fixed linear map [130], and then finally improved to any non-oblivious, non-linear embedding function by [131]. It is worth noting that the final construction is significantly different that that of previous works and does not use nearly orthogonal sparse vectors. Note that these lower bounds inherently rely on the assumption that the dot product of the embedding must approximately preserve the dot product, without any post-processing or decoding.

Speeding up JL for Sparse Vectors There have been a number of works on speeding up the runtime for embedding a collection of n sparse vectors using a JL map (while still embedding to $O(\log(n)/\varepsilon^2)$ dimensions). [121] demonstrates a distribution over sparse JL embedding matrices Π such that Πx takes $\tilde{O}(\|x\|_0/\varepsilon)$ time to compute, where $\|x\|_0$ denotes the number of non-zero entries of a vector x.

Distributional Embedding Lower Bounds and JL Recall that the standard JL lemma states that for any n vectors in $\mathbb{R}^d$, one can use a random Gaussian embedding to $O(\log(n)/\varepsilon^2)$ dimensions to guarantee the following maximum distortion bound with high probability: $(1-\varepsilon)\|x-y\|^2 \le \|Ax-Ay\|^2 \le (1+\varepsilon)\|x-y\|^2$ for all $x, y \in V$, where $|V| = n$. Furthermore, this embedding is also oblivious. While it is also known that the JL lemma is tight in the worst case, we emphasize that even our average case lower bounds towards Question (4) (see Section 4.1.4) are not implied by the existing JL lower bounds, even for the $p = 2$ case of Theorem 4.1.9. At a high level, this is because our hypothesis is much weaker (we only require a large fraction of the norm to be preserved, rather than all pairs). We elaborate below.

JL embedding lower bounds state that for large enough n, there exists a *specific* point set on n points such that *any map f* which preserves all pairwise distances must map to $\Omega(\log(n)/\varepsilon^2)$ dimensions [131]. The main difference between this lower bound and our lower bounds of Section 4.1.4 are that the hypotheses assumed are different. The JL upper bound guarantee implies approximate norm preservation for *every* pair of differences of points in a collection of n points, simultaneously and with high probability. Similarly, the JL lower bound assumes approximate norm preservation for *every* pair in a collection of n vectors. On the other hand, the folklore birthday paradox upper bound assumes a fixed sparse vector as input, whose norm it preserves with constant probability. Similarly, our lower bounds of Section 4.1.4 only assume approximate norm preservation only a constant fraction of the time across a uniform distribution of suitably chosen sparse vector inputs. Consequently, we have no term depending on the number of input vectors in the statement (both in the folklore upper bound and the lower bounds).

There is also the *distributional* JL lemma which is a random map from $\mathbb{R}^d \to \mathbb{R}^{O(1/\varepsilon^2)}$ preserving the norm of any fixed vector in $\mathbb{R}^d$ up to a multiplicative $1 \pm \varepsilon$ with probability at least 99% [120]. It is shown in [115] that the projection dimension of the distributional JL lemma is tight *information theoretically.* This ostensibly seems to imply our lower bounds (e.g. the $p = 2$ case of Theorem 4.1.9), for example if we parameterize the sparsity as $s = 1/\varepsilon$, seemingly rendering our lower bounds obsolete. But this is not the case! The information-theoretic lower bound proved in [115] relies on *dense* vectors in $\mathbb{R}^d$. This is not an artifact of the proof, but an inherent requirement to prove their information-theoretic lower bound: any such lower bound *cannot* use sparse vectors. This is because information-theoretically, there is already a smaller projection dimension from compressed sensing: we can encode any sparse vector in $\tilde{O}(s)$ dimensions. Parameterizing the sparsity by $s = 1/\varepsilon$, this implies an information-theoretic upper bound of $\tilde{O}(1/\varepsilon)$, demonstrating that the lower bound of [115] is not applicable to sparse vectors.

Compressed Sensing The field of compressed sensing studies decoding to recover encoded or compressed sparse vectors [75]. The restricted isometry property (RIP), introduced in [44], is a way to recover sparse vectors and focuses on bounding the RIP constant: the maximum distortion D of a linear map $A \in \mathbb{R}^{m\times}$ on all s-sparse vectors: $(\|x\|_p \leq \|Ax\|_p \leq D\|x\|_p$ for all x with $x \in \mathbb{R}^d$ and s-sparse. The mappings under matrices with good RIP constants can serve as the compressed representation. Furthermore, matrices with such properties automatically give dimensionality reduction for s-sparse vectors. The prior known bounds for m for RIP matrices with distortion $1 \pm \varepsilon$ are $O(s\log(d/s)/\varepsilon^2)$ for the $p = 1$ and 2 case [35, 44]. For other values of p, [209] gave bounds of the form $\tilde{O}(s^p)$ (see Table 4.1).

If the RIP constant is small enough, then applying a linear programming decoding via ℓ_1 minimization recovers the sparse vectors when compressed to the informational-theoretical optimal $O(s\log(d))$ dimensions, with a matching lower bound [138, 145]. There are other alternatives, e.g. based on CountSketch [57, 164], but all these decoders solve a complex optimization problem. However, these mappings do not serve as embeddings (e.g. the ℓ_p of the original vector is not approximated by the ℓ_p norm of the mapping).

Role of Sparsity in Machine Learning Sparse primitives such as sparse matrix multiplication is fundamental in a wide range of domains, such as graph analytics and scientific computing, and is used as iterative algorithms for sparse matrix factorization. Moreover, the field of deep learning often relies on faster sparse kernels to demonstrate speedups in practice, as it is a core operation in graph neural networks, transformers, and other architectures [65, 97]. On the theoretical side, there are also many works which seek to understand the role of sparsity in computational hardness for optimization problems such as sparse regression [64, 71, 82, 92, 95, 144, 157, 165].

4.4 Embedding for sparse vectors

We prove Theorem 4.4.2 in this section and give an embedding for sparse vectors. Later in Section 4.5, we give an embedding tailored to non-negative sparse vectors.

Our embedding is constructed in two parts. In the first part, we consider the birthday paradox linear mapping $A : \mathbb{R}^d \to \mathbb{R}^m$ given in Definition 4.0.1. It is a random mapping which preserves *all* ℓ_p norms of a *fixed* pair of sparse vectors with constant probability. Furthermore, it will have bounded expansion. Our final embedding is formed by stacking many independent copies of our base mappings. Our final embedding is the following.

Definition 4.4.1 (Final Embedding). *A (d, m, T) embedding $F : \mathbb{R}^d \to \mathbb{R}^{mT}$ is defined as follows. For every $1 \leq i \leq T$, let $A_i : \mathbb{R}^d \to \mathbb{R}^m$ be an independent copy of the random mapping of Definition 4.0.1. F is defined by stacking the matrices A_i on top of each other (note that F is linear), and dividing by $T^{1/p}$. In other words,*

$$Fx = \frac{1}{T^{1/p}} \cdot \bigoplus_{i=1}^{T} A_i x.$$

First we show the birthday paradox map has bounded distortion.

Lemma 4.4.1. *Let $A : \mathbb{R}^d \to \mathbb{R}^{100s^2/\delta}$ be the mapping of Definition 4.0.1. For any s-sparse vector $x \in \mathbb{R}^d$ independent of A,*

$$\Pr(\forall p,\ \|Ax\|_p = \|x\|_p) \geq 1 - \delta.$$

For every s-sparse vector x, $\|Ax\|_p \leq s^{1-1/p}\|x\|_p$ deterministically for all $p \geq 1$.

Proof. Consider the columns of A corresponding to the support of x. There are only at most s such columns. The probability that the non-zero rows of any two of these columns are the same is at most

$$\binom{2s}{2} \cdot \frac{1}{(100s^2/\delta)} < \delta.$$

Thus with probability at least $1-\delta$, the non-zero entries of Ax are exactly the non-zero entries of x. Under this event, the value of any ℓ_p norm (or any coordinate symmetric function) is preserved.

The second statement follows from the following fact: for any real numbers $x_1, \dots, x_k$ and $p \geq 1$, we have

$$(x_1 + \ldots + x_k)^p \leq k^{p-1}(|x_1|^p + \ldots + |x_k|^p) \tag{4.1}$$

due to convexity. □

We now prove the main theorem of the section. At a high level, we can compensate for the (bounded) distortion of the birthday paradox map by concatenating multiple independent copies.

Theorem 4.4.2. *Let $p \geq 1$ and $F : \mathbb{R}^d \to \mathbb{R}^{s^{p+2}\log(s)\log(d/\varepsilon)2^{O(p)}/\varepsilon^2}$ be a (d, m, T) embedding as stated in Definition 4.4.1 for $m = O(2^{O(p)}s^2/\varepsilon)$ and $T = O(\log(s)\log(d/\varepsilon)2^{O(p)}s^p/\varepsilon)$. Then*

$$\Pr\left(\forall\ s\text{-sparse } x \in \mathbb{R}^d,\ \|Fx\|_p = (1 \pm \varepsilon)\|x\|_p\right) \geq 0.99.$$

Proof. Let $A_1, \dots, A_T$ be the linear maps $A_i : \mathbb{R}^d \to \mathbb{R}^m$ for F where $T = \log(s)\log(d/\varepsilon)2^{O(p)}s^p/\varepsilon$ and $m = 2^{O(p)}s^2/\varepsilon$. Consider a fixed x for now. First, note that each i satisfies $\|A_ix\|_p = \|x\|_p$ with failure probability at most $s^2/m \leq \varepsilon$. Then with failure probability at most $\exp(-\Omega(T\varepsilon))$, at least $1 - 10\varepsilon$ fraction of the A_i's satisfy $\|A_ix\|_p = \|x\|_p$. Condition on this event. Then,

$$\|Fx\|_p^p \geq (1 - 10\varepsilon)\|x\|_p^p.$$

Now for $t = 2^k, k \geq 0$, let $\mathcal{E}_t^i$ be the event that at least t *and* at most $2t$ non-zero coordinates of x collide together under A_i. We have

$$\Pr(\mathcal{E}_t^i) \leq \binom{s}{t} \cdot \frac{1}{m^{t-1}} := p_t.$$

Since $t \geq 2$, we have

$$\begin{aligned}
p_t(2t)^{p-1} &= \binom{s}{t} \cdot \frac{(2t)^{p-1}}{m^{t-1}} \leq m \left(\frac{es}{tm}\right)^t (2t)^{p-1} \\
&\leq \frac{\varepsilon^{t-1}}{s^{t-2}} \cdot \frac{e^t (2t)^{p-1}}{2^{100p(t-1)}} \\
&\leq \varepsilon^{t-1} e^t \cdot \frac{(2t)^{p-1}}{2^{100p(t-1)}} \\
&\leq \varepsilon^{t-1} e^t \left(\frac{(2t)^{\frac{1}{t-1}}}{2^{100}}\right)^{p(t-1)} \\
&\leq \varepsilon / 2^t.
\end{aligned}$$

Now due to the independence of the different A_i's, the probability that at least $O\left(\frac{T\varepsilon}{(2t)^{p-1}} \cdot \frac{1}{\min(2^t, \log s)}\right)$ of the A_i's satisfy $\mathcal{E}_i^t$ is at most

$$\exp\left(-\Omega\left(\frac{T\varepsilon}{(2t)^{p-1}\log s}\right)\right) \leq \left(\frac{\varepsilon}{d}\right)^{1000s}$$

by the Chernoff bound and adjusting constants. Thus with failure probability at most $(\varepsilon/d)^{100s}$, for every t simultaneously, the number of A_i's that satisfy $\mathcal{E}_t^i$ is at most $O\left(\frac{T\varepsilon}{(2t)^{p-1}} \cdot \frac{1}{\min(2^t, \log s)}\right)$. Conditioning on this, we have

$$\|Fx\|_p^p \leq (1 - 10\varepsilon)\|x\|_p^p + \sum_{t \geq 2} (2t)^{p-1} O\left(\frac{\varepsilon}{(2t)^{p-1}} \cdot \frac{1}{\min(2^t, \log s)}\right) \|x\|_p^p,$$

where the outer $(2t)^{p-1}$ factor is due to the same argument as in Equation 4.1. Now since there are at most $O(\log s)$ terms in the above sum,

$$\sum_{t \geq 2} (2t)^{p-1} O\left(\frac{\varepsilon}{(2t)^{p-1}} \cdot \frac{1}{\min(2^t, \log s)}\right) \leq \sum_{t \geq 2} (2t)^{p-1} O\left(\frac{\varepsilon}{(2t)^{p-1}} \cdot \left(\frac{1}{2^t} + \frac{1}{\log s}\right)\right) = O(\varepsilon),$$

and thus with failure probability at most $(\varepsilon/d)^{100s}$,

$$\|Fx\|_p^p \leq (1 - 10\varepsilon)\|x\|_p^p + O(\varepsilon)\|x\|_p^p \leq (1 + O(\varepsilon))\|x\|_p^p.$$

So far we have considered a single fixed x. We now apply this reasoning to a large collection via the union bound. Let $\mathcal{N}$ be an γ-net in ℓ_p norm of all s-sparse vectors z in $\mathbb{R}^d$ satisfying $\|z\|_p = 1$ with $\gamma = \varepsilon^2/d^2$. We have that $|\mathcal{N}| \leq (d/\varepsilon)^{50s}$. By our choice of T and the above calculations, we know that

$$(1 - 10\varepsilon)\|z\|_p^p \leq \|Fz\|_p^p \leq (1 + 10\varepsilon)\|z\|_p^p$$

for all $z \in \mathcal{N}$ simultaneously with probability at least 0.99. By taking the $1/p$th power, we also have

$$\forall z \in \mathcal{N},\ (1-10\varepsilon)\|z\|_p \le \|Fz\|_p \le (1+10\varepsilon)\|z\|_p \tag{4.2}$$

where we have used the fact that $1+10\varepsilon \ge (1+10\varepsilon)^{1/p}$ and similarly, $(1-10\varepsilon)^{1/p} \ge 1-10\varepsilon$ since $p \ge 1$ and $\varepsilon \in (0,1)$. We now extend the guarantees of Equation (4.2) for *all* s-sparse $y \in \mathbb{R}^d$.

Since F is linear, we can without loss of generality assume $\|y\|_p = 1$ by scaling. Pick $z \in \mathcal{N}$ that satisfies $\|z-y\|_p \le \gamma$. We know $\|z\|_p = 1$ and $|\|Fz\|_p - 1| \le 10\varepsilon$. Then

$$|1 - \|Fy\|_p| \le |\|Fz\|_p - \|Fy\|_p| + 10\varepsilon \le \|F(z-y)\|_p + 10\varepsilon,$$

and

$$\|F(z-y)\|_p^p \le \gamma^p \max_i \|A_i\|_p^p \le \gamma^p \max_i(\|A_i\|_\infty^{p-1}\|A_i\|_1) \le d\gamma^p$$

(where we are using the induced matrix norm). Hence,

$$|\|Fy\|_p - 1| = |\|Fy\|_p - \|y\|_p| \le 10\varepsilon + d^{1/p}\gamma \le 11\varepsilon = 11\varepsilon\|y\|_p,$$

completing the proof. The theorem statement follows by adjusting the value of ε. □

4.5 Embedding for Non-negative Sparse Vectors

In this section we present our embeddings for non-negative sparse vectors and prove Theorem 4.5.2.

Our embedding is again constructed in two parts. In the first part, we introduce a 'base' mapping $f : \mathbb{R}^d \to \mathbb{R}^m$. It will be a random mapping which preserves *all* ℓ_p norms of a *fixed* pair of non-negative sparse vectors with constant probability. Crucially, it will be non-expanding. Our final embedding is formed by stacking many independent copies of our base mapping.

Definition 4.5.1 (Base mapping)**.** *We define a mapping $f : \mathbb{R}^d \to \mathbb{R}^m$. Pick a uniformly random function from $h : [d] \to [m]$. For every $i \in [m]$, let $S_i = \{j \in [d] \mid h(j) = i\}$. We define $f(x) \in \mathbb{R}^m$ as follows. For every $i \in [m]$,*

$$f(x)_i = \begin{cases} \max(\{x_j \mid j \in S_i\}) & \text{if } S_i \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

Our final embedding is the following.

Definition 4.5.2 (Final Embedding)**.** *A non-negative (d, m, T) embedding $F : \mathbb{R}^d \to \mathbb{R}^{mT}$ is defined as follows. For every $1 \le i \le T$, let $f_i : \mathbb{R}^d \to \mathbb{R}^m$ be an independent copy of the random mapping of Definition 4.5.1. Then for any $x \in \mathbb{R}^d$,*

$$F(x) = \bigoplus_{i=1}^{T} f_i(x).$$

The key properties of the base mapping are proved below. As discussed in Section 4.2, the non-expansion property is particularly crucial.

Theorem 4.5.1. *Let $f : \mathbb{R}^d \to \mathbb{R}^m$ be a random mapping of Definition 4.5.1. It satisfies the following:*

1. *$f(0) = 0$ deterministically for all values $m \geq 1$.*
2. *For any pair of non-negative s-sparse vectors $x, y \in \mathbb{R}^d$ (both independent of f), if we take $m = \Omega(s^2/\delta)$,*
$$\Pr(\forall p,\ \|f(x) - f(y)\|_p = \|x - y\|_p) \geq 1 - \delta.$$
3. *For every pair of non-negative vectors x and y (not necessarily sparse) and any embedding dimension m, $\|f(x) - f(y)\|_p \leq \|x - y\|_p$ deterministically for all $p \geq 1$.*

Proof. The first property follow from the definition of f. Let $m = 100s^2/\delta$ and h be the uniformly random function from $[d] \to [m]$ that constitutes f.

Let x and y be two fixed s-sparse vectors in $\mathbb{R}^d$. Proving the second condition relies solely on the fact that h is likely to separate all the non-zero coordinates of x and y. Indeed, the union of their supports is of size at most $2s$. Under h, the probability that some two domain elements in their union collide is at most
$$\binom{2s}{2} \cdot \frac{1}{(100s^2/\delta)} < \delta.$$
This event means that very coordinate j in the union of the supports of x and y is mapped to a unique coordinate in $[m]$, and thus for every p,
$$\|x - y\|_p^p = \sum_{j=1}^{d} |x_i - y_i|^p = \sum_{i=1}^{m} |f(x)_i - f(y)_i|^p = \|f(x) - f(y)\|_p^p,$$
proving the second condition.

The third condition crucially relies on the fact that we are using the 'max' operation to define f. For any pair of non-negative vectors x and y, we have
$$\|f(x) - f(y)\|^p = \sum_{i=1}^{m} |f(x)_j - f(y)_j|^p = \sum_{i=1}^{m} |\max(\{x_j \mid h(j) = i\}) - \max(\{y_j \mid h(j) = i\})|^p.$$
It suffices to prove the following: if $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ are non-negative real numbers then
$$|\max(a_1, \ldots, a_k) - \max(b_1, \ldots, b_k)| \leq \max_t |a_t - b_t|.$$
This is because assuming the claim, we have
$$\sum_{i=1}^{m} |\max(\{x_j \mid h(j) = i\}) - \max(\{y_j \mid h(j) = i\})|^p \leq \sum_{i=1}^{m} \max_{j|h(j)=i} |x_j - y_j|^p \leq \sum_{j=1}^{d} |x_i - y_i|^p,$$

since h maps every coordinate in $[d]$ to only one coordinate in $[m]$.

Now to prove the claim, assume without loss of generality that $a_{t'} = \max(a_t) \geq \max_t(b_t)$. Then $|a_{t'} - \max_t(b_t)| \leq |a_{t'} - b_{t'}|$ since in the real line, we have the ordering $b_{t'} \leq \max_t(b_t) \leq a_{t'}$. □

Building upon Theorem 4.5.1, we give an embedding which approximately preserves all pairwise distances between points in a dataset of non-negative sparse vectors.

Theorem 4.5.2 (Embedding for non-negative sparse vectors). *Let $F : \mathbb{R}^d \to \mathbb{R}^{mT}$ be a non-negative (d, m, T) embedding for*

$$m = O\left(\min\left(\frac{s^2}{\varepsilon}, \frac{s}{\varepsilon^2}\right)\right), \quad T = O\left(\frac{\log(n)}{\varepsilon}\right)$$

as stated in Definition 4.5.2. Let $X \subset \mathbb{R}^d$ be a dataset of n non-negative s-sparse vectors (which is independent of F). We have

1. $$\Pr\left(\forall p \geq 1, \forall x, y \in X, \frac{\|F(x) - F(y)\|_p^p/T}{\|x - y\|_p^p} \in 1 \pm \varepsilon\right) \geq 1 - 1/\text{poly}(n).$$

2. *For the ℓ_∞ norm, it suffices to take $m = O(s)$ and $T = O(\log n)$ and guarantee that*
$$\Pr\left(\forall x, y \in X, \|F(x) - F(y)\|_\infty = \|x - y\|_\infty\right) \geq 1 - 1/\text{poly}(n).$$

3. *Additionally for the ℓ_∞ norm, it suffices to take $m = 1$ and $T = 1$ and guarantee that*
$$\Pr\left(\forall x, y \in X, \frac{\|F(x) + F(y)\|_\infty}{\|x + y\|_\infty} \in [1, 2]\right) \geq 1 - 1/\text{poly}(n).$$

Proof. We first prove part (1) in the case $m = O(s^2/\varepsilon)$. Note that

$$F = \bigoplus_{k=1}^{T} f_k$$

where each $f_k : \mathbb{R}^d \to \mathbb{R}^{O(s^2/\varepsilon)}$ is the mapping of Definition 4.5.1 and $T = O(\log(n)/\varepsilon)$. Now consider an arbitrary fixed pair $x, y \in X$. For this pair, we know every f_k is non-expanding and satisfies $\|f_k(x) - f_k(y)\|_p^p = \|x - y\|_p^p$ for all $p \geq 1$ simultaneously with probability at least $1 - \varepsilon/10$. Since f_k's are independent, the number of indices k for which exact equality holds is a binomial random variable. A standard concentration bound (Lemma 4.9.1) shows that $\|f_k(x) - f_k(y)\|_p^p = \|x - y\|_p^p$ for at least a $(1 - \varepsilon/2)T$ of the indices with failure probability at most $\exp(-\Omega(T\varepsilon)) = 1/\text{poly}(n)$. If this is the case, we have

$$\sum_k \|f_k(x) - f_k(y)\|_p^p \geq (1 - \varepsilon/2)T \cdot \|x - y\|_p^p$$

and by the non-expanding property of f_k, we also have

$$\sum_k \|f_k(x) - f_k(y)\|_p^p \le T \cdot \|x - y\|_p^p.$$

The first part of the theorem for the $m = O(s^2/\varepsilon)$ case then follows by a union bound over all $\Theta(n^2)$ pairs.

We now show that $m = O(s/\varepsilon^2)$ also suffices. Consider the random variable $\|f_1(x) - f_1(y)\|_p^p$. We know deterministically $\|f_1(x) - f_1(y)\|_p^p \le \|x - y\|_p^p$ no matter what. For a non-zero coordinate z_i of $z := x - y$, let t_i denote the indicator random variable for the event that i does not collide with any of the other non-zero coordinates of z. By our choice of m, we know that $\mathbb{E}[t_i] \ge 1 - \varepsilon^2$ since the sparsity of z is $O(s)$, as it is the difference of two s-sparse vectors. Thus,

$$\mathbb{E}[\|f_1(x) - f_1(y)\|_p^p] \ge (1 - \varepsilon^2)\|x - y\|_p^p,$$

so by Markov's inequality,

$$0 \le \mathbb{E}\left[\|x - y\|_p^p - \|f_1(x) - f_1(y)\|_p^p\right] \le \varepsilon \|x - y\|_p^p$$

with probability at least $1 - \varepsilon$. Now by applying the Chernoff bound as before, with failure probability at most $\exp(-\Omega(T\varepsilon)) = 1/\text{poly}(n)$, we have $\|f_k(x) - f_k(y)\|_p^p \ge (1 - \varepsilon)\|x - y\|_p^p$ for at least a $(1 - \varepsilon/2)T$ of the indices. The proof finishes identically as before.

The ℓ_∞ case can be handled as follows. For a fixed pair x, y, suppose that the first coordinate witnesses their ℓ_∞ norm. If we take $m = O(s)$, then the first coordinate does not collide with any other coordinate with probability at least 99% (there maybe collisions among other coordinates). Thus with large constant probability, a fixed base mapping f of Definition 4.5.1 certifies that $\|f(x) - f(y)\|_\infty \ge \|x - y\|_\infty$. But we always have $\|f(x) - f(y)\|_\infty \le \|x - y\|_\infty$ deterministically. Thus with $O(\log n)$ repetitions, every pair x, y will have at least one copy of the base mapping which ensures that the ℓ_∞ distance is *exactly* preserved.

For the third part of the theorem, note that since the coordinates are non-negative and $F(x) \in \mathbb{R}$ is just the maximum coordinate, we trivially have $\|F(x) + F(y)\|_\infty \ge \|x + y\|_\infty$. In the other direction, we claim that $\|f(x) + f(y)\|_\infty \le 2\|x + y\|_\infty$ always holds deterministically where f is our base mapping of Definition 4.5.1. Indeed, it suffices to show that for any non-negative real numbers $a_1, \ldots, a_r$ and $b_1, \ldots, b_r$, we always have

$$\max(a_1, \ldots, a_r) + \max(b_1, \ldots, b_r) \le 2 \max_{t \in [r]}(a_t + b_t). \tag{4.3}$$

This is seen to be true by just taking $t = \arg\max a_t$ (w.l.o.g. $\max(a_t) \ge \max(b_t)$). This completes the proof. □

The *additive* guarantees of our mapping F also extends to general ℓ_p norms, with an additional overhead of $2^{O(p)}$.

Corollary 4.5.3. *Let $p \geq 1$ and $F : \mathbb{R}^d \to \mathbb{R}^{s^2 2^{O(p)} \log(n)/\varepsilon^2}$ be a non-negative $(d, s^2 2^{O(p)}/\varepsilon, T)$ embedding for $T = 2^{O(p)} \log(n)/\varepsilon$ as stated in Definition 4.5.2. Let $X \subset \mathbb{R}^d$ be a dataset of n non-negative s-sparse vectors (which is independent of F). We have*

$$\Pr\left(\forall x, y \in X, \frac{\|F(x) + F(y)\|_p^p/T}{\|x+y\|_p^p} \in 1 \pm \varepsilon\right) \geq 1 - 1/\mathrm{poly}(n).$$

Proof. The proof is very similar to that of Theorem 4.5.2 so we only highlight the differences. For the parameters, we take $f_k : \mathbb{R}^d \to \mathbb{R}^m$ for $m = s^2 2^{O(p)}/\varepsilon$ and $T = \log(n) 2^{O(p)}/\varepsilon$. Let $\gamma = s^2/m$. First, for any fixed pair $x, y \in X$ we have $\|f_k(x) + f_k(y)\|_p^p = \|x+y\|_p^p$ holds for at least $1 - \gamma$ fraction of k's with failure probability at most $1/\mathrm{poly}(n)$. Thus we have

$$\sum_k \|f_k(x) + f_k(y)\|_p^p \geq (1-\gamma)T \cdot \|x+y\|_p^p.$$

To bound the other direction, we also always have $\|f_k(x) + f_k(y)\|_p^p \leq 2^p \|x+y\|_p^p$ for every k from Eq. 4.3. This means that

$$\sum_k \|f_k(x) + f_k(y)\|_p^p \leq (1-\gamma)T \cdot \|x+y\|_p^p + \gamma T 2^p \|x+y\|_p^p.$$

Recalling the value of γ finishes the proof as in Theorem 4.5.2. □

If we limit the entries of the sparse vectors to be from a discrete set, then we can extend our non linear map towards for general vectors as well.

Theorem 4.5.4. *Let $p \geq 1$ and $F : \mathbb{R}^d \to \mathbb{R}^{s^2 \Delta^{O(p)} \log(n)/\varepsilon^2}$ be a $(d, s^2 \Delta^{O(p)}/\varepsilon), T)$ embedding for $T = \log(n)\Delta^{O(p)}/\varepsilon$ as stated in Definition 4.5.2. Let $X \subset \mathbb{R}^d$ be a dataset of n s-sparse vectors with entries in the discrete set $\{-\Delta, \ldots, \Delta\}$. We have*

$$\Pr\left(\forall x, y \in X, \frac{\|F(x) - F(y)\|_p^p}{\|x-y\|_p^p} \in [(1-\varepsilon)T, T]\right) \geq 1 - 1/\mathrm{poly}(n).$$

Proof. We again consider

$$F = \bigoplus_{k=1}^{T} f_k$$

where each $f_k : \mathbb{R}^d \to \mathbb{R}^m$ for $m = s^2(2\Delta)^p/\varepsilon$ is the mapping of Definition 4.5.1 and $T = O(\log(n)(2\Delta)^p/\varepsilon)$. We can check that the first two properties of f in Theorem 4.5.1 hold for arbitrary s-sparse vectors. However, the third property crucially relies on non-negative entries. Nevertheless, we can obtain the following variant, assuming the entries are in a discrete set: for every pair of vectors $x, y \in \{-\Delta, \ldots, \Delta\}^d$, we have $\|f_k(x) - f_k(y)\|_p \leq \Delta \|x - y\|_p$. Indeed, as in the proof of Theorem 4.5.1, it suffices to prove the following: if $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ are all in $\{-\Delta, \ldots, \Delta\}$, then

$$|\max(a_1, \ldots, a_k) - \max(b_1, \ldots, b_k)| \leq 2\Delta \cdot \max_t |a_t - b_t|.$$

And this holds because if the RHS is 0, then so is the LHS. Otherwise, the RHS is at least $2\Delta \cdot 1$ and the LHS is always bounded by 2Δ.

Equipped with this, we similarly have that with failure probability at most $\exp(-\Omega(T\gamma)) = 1/\text{poly}(n)$, $\|f_k(x) - f_k(y)\|_p^p = \|x-y\|_p^p$ for at least $(1-\gamma/2)T$ fraction of the indices k where $\gamma = \varepsilon/(2\Delta)^p$. If this is the case, then again

$$\sum_k \|f_k(x) - f_k(y)\|_p^p \geq (1-\gamma/2)T \cdot \|x-y\|_p^p$$

and by the bounded-expanding property of f_k, we also have

$$\sum_k \|f_k(x) - f_k(y)\|_p^p \leq (1-\gamma/2)T \cdot \|x-y\|_p^p + \gamma T(2\Delta)^p \|x-y\|_p^p.$$

Putting everything together, we have $\sum_k \|f_k(x) - f_k(y)\|_p^p \in (1 \pm \varepsilon/2)T \cdot \|x-y\|_p^p$. And the final result follows from union bounding over all $\Theta(n^2)$ pairs, as desired. □

As a corollary, we can embed ℓ_∞ into a smaller dimensional space in $\ell_{O(\log(s)/\varepsilon)}$.

Corollary 4.5.5. *Let $p = O(\log(s)/\varepsilon)$. Let X be a set of s-sparse vectors in $\mathbb{R}^d$ (arbitrary entries). There is an embedding of X into ℓ_p^m which preserves all pairwise ℓ_∞ distances up to $1 \pm \varepsilon$ factor for the following values of m:*

- *If we use the embedding of [209], then $m = s^{O(\varepsilon^{-1}(\log\log d + \log s))}/\varepsilon^2$.*
- *If we use the embedding of Theorem 4.4.2, then $m = \log(s)\log(d/\varepsilon)s^{O(\varepsilon^{-1}\log s)}/\varepsilon^2$.*
- *If $|X| \subset \{-O(1), \dots, O(1)\}$, then the embedding of Theorem 4.5.4 gives $m = \log(|X|)s^{O(1/\varepsilon)}/\varepsilon^2$.*

Proof. The proof follows by noting that in s dimensions, the ℓ_∞ norm is $1 \pm \varepsilon$ approximated by the ℓ_p norm for $p = O(\log(s)/\varepsilon)$ (Lemma 4.9.2), and plugging in this value of p in the respective theorems. □

4.6 Dimensionality Reduction Lower Bounds for Non-Negative Vectors

In this section, we provide lower bounds for dimensionality reduction for sparse vectors under various hypotheses. As stated in Section 4.1, together they demonstrate that our non-negative sparse embedding of Theorem 4.5.2 is optimal in many natural ways. See Section 4.1 for an overview.

We begin by showing that a linear map cannot have the same guarantees as Theorem 4.5.2.

Theorem 4.1.4. *Let S be the set of all 10-sparse vectors in $\mathbb{R}^d$ with all non-zero coordinates being equal to 1. Let $A : \mathbb{R}^d \to \mathbb{R}^m$ be a matrix such that*

$$\frac{1}{2}\|x\|_\infty \leq \|Ax\|_\infty \leq \frac{3}{2}\|x\|_\infty \quad \forall x \in S.$$

Then $m = \Omega(d)$.

Proof. Suppose for the sake of contradiction that $m < d/100$. By considering the basis vectors, every column of A must have an entry with absolute value at least $1/2$. Then there must exist a row r of A which has at least $d/m \geq 100$ such entries. At least 50 of such entries in row r must be of the same sign. Let x be an indicator vector for the column of 10 these entries in row r. Note that $x \in S$ and $|(Ax)_r| \geq 5$ which implies $\|Ax\|_\infty \geq 5$, contradicting our assumption on A. Thus, $m \geq d/100 = \Omega(d)$, as desired. □

The following theorem states that preserving the norms of the sum of vectors is impossible a factor of $2 - \varepsilon$ for any $\varepsilon > 0$ in ℓ_∞, even if we only restrict to non-negative sparse vectors. Note that Theorem 4.5.2 preserves the norms of the sums up to a factor of 2.

Theorem 4.1.6. *Let e_i be the ith basis vector in $\mathbb{R}^d$. Consider the set $S = \{e_i\} \cup \{0\}$ of $d+1$ vectors. Suppose $f : S \to \mathbb{R}^m$ be an arbitrary mapping which satisfies*

$$\|x + y\|_\infty \leq \|f(x) + f(y)\|_\infty \leq (2 - \varepsilon)\|x - y\|_\infty \quad \forall x, y \in S$$

for any $\varepsilon > 0$. Then $m \geq d$.

Proof. Suppose for the sake of contradiction that $m < d$. First note that $f(0)$ must be the all 0's vector by taking $x = y = 0$. Then taking $x = e_i$ and $y = 0$ implies that $\|f(e_i)\|_\infty \in [1, 2 - \varepsilon]$ for all i. Again label such coordinates of $f(e_i)$ that lie in this range as 'large.' If $m < d$, then there exists i and j such that $f(e_i)$ and $f(e_j)$ have the same large index by Pigeonhole. We know that $\|f(e_i) + f(e_j)\|_\infty \in [1, 2 - \varepsilon]$ by our hypothesis. Now if the large entries of $f(e_i)$ and $f(e_j)$ have the same sign, then their sum in absolute value is at least 2, which cannot happen from the above observation. On the other hand, if they have different signs, then the largest sum (in absolute value) that these entries can add to is at most $1 - \varepsilon$ (either from $2 - \varepsilon + (-1)$ or $-(2 - \varepsilon) + 1$), which also cannot happen. These cases are exhaustive which means our assumption $m < d$ cannot hold, and we must have $m \geq d$. □

The following theorem shows that any mapping f also cannot be 'too smooth'. Note that in our mapping, we use the maximum function, which is not differentiable.

Theorem 4.1.5. *Suppose $f : \mathbb{R}^d \to \mathbb{R}^m$ is such that $f(x) = (f_1(x), \ldots, f_m(x))$ where each f_i is twice differentiable with continuous second partial derivatives. Let S be the set of all 10-sparse vectors in $\mathbb{R}^d$ with all non-zero coordinates being equal to 1. Suppose f satisfies*

$$0.99\|rx\|_\infty \leq \|f(rx)\|_\infty \leq 1.01\|rx\|_\infty \quad \forall r > 0, \forall x \in S.$$

Then $m = \Omega(d)$.

Proof. Note that $f_i(0) = 0$ by taking $r \to 0$. Since the second partial derivatives of all f_i are continuous, they are bounded in magnitude in the compact set $[0,1]^d$. Let T be such an upper bound which holds for all i. Now we let $r \ll 1$ be a sufficiently small value which will be determined shortly. For any fixed f_i, Taylor's theorem for multivariate functions[2] implies that for any $y \in [0,r]^d$ which is $O(1)$-sparse,

$$|f_i(y) - \langle \nabla f_i(0), y \rangle| \leq T\,(y_1 + \ldots + y_d)^2 \leq O(Tr^2)$$

for every i. Let $A : \mathbb{R}^d \to \mathbb{R}^m$ be the matrix with a_i as it's rows. The above inequality implies that

$$\|f(y) - Ay\|_\infty \leq O(Tr^2).$$

Thus if $r > 0$ sufficiently small, z is any vector such that $\|z\|_\infty = 1$, and

$$0.99\|rz\|_\infty \leq \|f(rz)\|_\infty \leq 1.01\|rz\|_\infty,$$

then we must also have

$$\frac{1}{2}\|rz\|_\infty \leq \|A(rz)\|_\infty \leq \frac{3}{2}\|rz\|_\infty.$$

But since A is linear, this actually implies that

$$\frac{1}{2}\|r'z\|_\infty \leq \|A(r'z)\|_\infty \leq \frac{3}{2}\|r'z\|_\infty$$

for all $r' > 0$. Putting everything together, (letting z be the vectors in the hypothesis of the theorem statement), implies that

$$\frac{1}{2}\|z\|_\infty \leq \|Ax\|_\infty \leq \frac{3}{2}\|z\|_\infty \quad \forall z \in S.$$

Then Theorem 4.1.4 implies that $m = \Omega(d)$, as desired. □

The next lower bound of this section demonstrates that the non-negative hypothesis is crucial. If we drop the non-negativity constraint, then any map cannot satisfy the guarantees promised by Theorem 4.5.2.

Theorem 4.1.7. *Let e_i be the ith basis vector in $\mathbb{R}^d$. Consider the set $S = \{\pm e_i\} \cup \{0\}$ of $2d+1$ vectors. Suppose $f : S \to \mathbb{R}^m$ be an arbitrary mapping which satisfies*

$$0.9\|x-y\|_\infty \leq \|f(x) - f(y)\|_\infty \leq 1.1\|x-y\|_\infty \quad \forall x,y \in S,$$

$$\|x+y\|_\infty \leq \|f(x) + f(y)\|_\infty \leq C\|x+y\|_\infty \quad \forall x,y \in S$$

for any $C \geq 1$. Then $m \geq d$.

[2] https://en.wikipedia.org/wiki/Taylor's_theorem#Taylor's_theorem_for_multivariate_functions

Proof. Suppose for the sake of contradiction that $m \le d-1$. We first show that $f(e_i) = -f(-e_i)$ for all i. Indeed, $e_i + (-e_i) = 0$ so we must have $\|f(e_i)+f(-e_i)\|_\infty = 0 \implies f(e_i) = -f(-e_i)$. We also have $f(0) = 0 \in \mathbb{R}^m$, also using the second relation. This means that $\|f(\pm e_i)\|_\infty \in [0.9, 1.1]$ for all i by using the first relation with $x = \pm e_i$ and $y = 0$.

For a vector $v \in S$, call the indices where $\|f(v)\|_\infty \in [0.9, 1.1]$ *large entries*. Our goal is to show that most vectors in S need to have distinct large entries. This will imply that m must be large since $|S|$ is large.

Indeed, if $m \le d-1$, then there must exists some $i \ne j$ such that $f(e_i)$ and $f(e_j)$ share the same large entry index by Pigeonhole. Then either $f(e_j)$ or $f(-e_j) = -f(e_j)$ also has the *opposite sign* as the large entry of $f(e_i)$. So at least in one case, we can find two vectors $x, y \in S$ such that $\|f(x) - f(y)\|_\infty \ge 2 \cdot 0.9 = 1.8$. However, since $i \ne j$, we know that $\|x - y\|_\infty = 1$, contradicting the first relation. Hence, $m \ge d$, as desired. □

Finally, the last lower bound demonstrates that any map with weaker guarantees than that of Theorem 4.5.2 must map to $\tilde{\Omega}(s)$ dimensions. Note that Theorem 4.5.2 satisfies the hypothesis of the theorem bellow with $C = O(1/\log n)$ for a dataset of n non-negative s-sparse vectors.

Theorem 4.6.1. *Suppose $d > 2s$ and let $S = \left\{\sum_{i=1}^s e_i, \sum_{i=s+1}^{2s} e_i, 0\right\}$. Suppose $f : S \to \mathbb{R}^m$ be an arbitrary mapping which satisfies*

$$\|f(x) - f(y)\|_\infty \le \|x - y\|_\infty \quad \forall x, y \in S,$$

$$C\|x - y\|_1 \le \|f(x) - f(y)\|_1 \quad \forall x, y \in S,$$

for any $C > 0$. Then $m \ge Cs$.

Proof. By shifting, we can assume $f(0) = 0$. Let $v_1 = \sum_{i=1}^s e_i$ and $v_2 = \sum_{i=s+1}^{2s} e_i$. Then the first relation implies that all coordinates of v_1 and v_2 are bounded by 1 in absolute value. The second relation them implies that

$$2Cs \le \|f(v_1) - f(v_2)\|_1 \le \|f(v_1)\|_1 + \|f(v_2)\|_1 \le 2m,$$

yielding $m \ge Cs$. □

4.7 Average Case Lower Bounds for Embedding General Sparse Vectors

The goal of this section is to prove lower bounds showing that the birthday paradox map of Definition 4.0.1 is optimal in many settings. The point set we use are generated randomly from the following distribution as discussed in Section 4.2.

Definition 4.2.1. *Let $\text{Unif}_{t,r}$ be a distribution over t-sparse vectors defined as follows. To generate $x \sim \text{Unif}_{t,r}$,*

1. *First pick t coordinates uniformly at random to be the support.*

2. *The non-zero coordinates of x are i.i.d. Gaussians with variance r.*

Our first result is to show that any linear map with the same average case guarantees as the birthday paradox map must map to $\Omega(s^2)$ dimensions. Note that the birthday paradox map satisfies the hypothesis of the theorem statement below up to constant factors (by mapping to Cs^2 dimensions for a sufficiently large constant C). This is because any fixed set of coordinates of size at most s hashes to unique buckets under the map with probability say at least 99% (by picking large enough C). If this is the case, then it does not matter what entries we put in this set of coordinates.

Theorem 4.7.1. *Let $A : \mathbb{R}^{s^2} \to \mathbb{R}^m$ be a linear map and $p \geq 2$ be an even integer. If A is such that for any $1 \leq t \leq s$,*

$$\Pr_{x \sim \mathrm{Unif}_t} (\|Ax\|_p = \|x\|_p) \geq 0.99,$$

then $m \geq s^2/1000$.

Proof. Suppose for the sake of contradiction that $m < s^2/1000$. The proof is roughly divided into three parts. The first part shows that A^TA has many 'non-zero' coordinates. The second part shows that a random sparse vector u (as chosen in the hypothesis of the theorem statement) has 'many' pairs of coordinates u_i and u_j such that the corresponding (i,j) entry in A^TA is also non-zero. The last part then shows that the prior result implies that A *does not* (approximately) preserve the norm of u with a large constant probability, contradicting the assumption of the theorem. The last part relies on the fact that the zero set of a non-zero polynomial has measure 0.

Part #1. Let $A_1, \ldots, A_{s^2}$ denote the columns of A. Let v be a random vector chosen from Unif_1, and let i be it's support with v_i the corresponding non-zero entry. Then $\|Av\|_p^p = \|A_i\|_p^p \cdot v_i^p$ and $\|v\|_p^p = v_i^p$. Then the hypothesis implies that

$$\Pr(|\|A_i\|_2^2 - 1| \leq 0.0001) \geq 0.99. \tag{4.4}$$

(Note that we deliberately work with a weaker hypothesis since it will be more useful to us later on). Thus, a 0.99 fraction of the columns of A have Euclidean norm in the range $[0.999, 1.001]$. Let A_X be the restriction to such columns. Note that all diagonal entries of $A_X^TA_X$ are at in the range $[0.999, 1.001]$ and all entries are bounded by 1.001 in absolute value (via Cauchy-Schwarz on the columns of A_X). For $i \geq 1$, we let ℓ_i denote the number of *non-diagonal* entries of $A_X^TA_X$ whose absolute values are in $(2^{-i}, 2^{-i+1}]$ and ℓ_0 denote the rest of the non-diagonal entries (with absolute values in $(1, 1.001]$). We show the following claims hold.

a) $\|A_X\|_F^2 \geq (0.99)^2 s^2$,

b) $\|A_X^T A_X\|_F^2 \geq \frac{(.99)^4 s^4}{m}$,

c) $\sum_{i\geq 0} \ell_i 2^{-2i+2} \geq \frac{(.99)^4 s^4}{m} - 1.001 s^2$.

Claim (a) readily follows from inequality (4.4). To show (b), note that singular values of $A_X^T A_X$ are the squared singular values of A_X so $\|A_X^T A_X\|_F^2 = \sum_{i=1}^m \sigma_i(A_X)^4$ (since the rank of A is at most m). By Cauchy–Schwarz,

$$m \cdot \|A_X^T A_X\|_F^2 = m \sum_i \sigma_i(A_X)^4 \geq \left(\sum_i \sigma_i(A_X)^2 \right)^2 = \|A_X\|_F^4 \geq (.99)^4 s^4.$$

To show (c), note that all diagonal entries of $A_X^T A_X$ are at in the range $[0.999, 1.001]$. Claim (c) then follows from using the lower bound of Claim (b) since there are at most s^2 diagonal entries.

Part #2. Now let u be a vector drawn from Unif_s and let T be its support. The hypothesis of the theorem states that

$$\Pr(\|Au\|_p = \|u\|_p) \geq 0.99. \tag{4.5}$$

Our goal is to demonstrate a contradiction by showing $\Pr(\|Au\|_p \neq \|u\|_p) \geq 0.02$, which contradicts (4.5).

Let u_X be the restriction of u to the coordinates in X (i.e., only keep the coordinates in X). Consider $u_X u_X^T$. By our choice of u, the non-zero entries of $u_X u_X^T$ lie on a random principal submatrix of size $|T \cap X| \times |T \cap X|$. We show that with a sufficiently large constant probability, both $u_X u_X^T \in \mathbb{R}^{|X|\times|X|}$ and $A_X^T A_X \in \mathbb{R}^{|X|\times|X|}$ have a non-zero value in 'many' shared entries.

Towards this end, let Y_{ij} be the indicator variable for the entry (i,j) in $u_X u_X^T$ being non-zero and let s_{ij} denote the *squared* (i,j) entry of $A_X^T A_X$. We know

$$\mathbb{E}[Y_{ij}] = \frac{s(s-1)}{s^2(s^2-1)} = \frac{1}{s(s+1)}.$$

Finally, let $Z = \sum_{i,j} s_{ij} Y_{ij}$. Recalling our partitions ℓ_k that we defined earlier and Claim (c), we have

$$\mathbb{E}[Z] = \sum_{i,j} s_{ij}\, \mathbb{E}[Y_{ij}] \geq \frac{1}{s(s+1)} \sum_k \ell_k 2^{-2k} \geq \frac{1}{4s(s+1)} \left(\frac{(.99)^4 s^4}{m} - 1.001 s^2 \right) > 200.$$

We want to show Z concentrates well around its mean. Towards this end, we bound $\mathbb{E}[Z^2]$. We have

$$Z^2 = \left(\sum_{i,j} s_{ij} Y_{ij} \right)^2 = \sum_{i,j} s_{ij}^2 Y_{ij} + \sum_{i,j,j'} s_{ij} s_{ij'} Y_{ij} Y_{ij'} + \sum_{i,j,i',j'} s_{ij} s_{i'j'} Y_{ij} Y_{i'j'}.$$

Since $s_{ij}^2 \leq 1.001$, the first term can be bounded as

$$\mathbb{E}\left[\sum_{i,j} s_{ij}^2 Y_{ij}\right] \leq \frac{1.001}{s(s+1)} \sum_{i,j} s_{ij} = 1.001\,\mathbb{E}[Z].$$

Similarly, the third term can be bounded as

$$\begin{aligned}\mathbb{E}\left[\sum_{i,j,i',j'} s_{ij}s_{i'j'}Y_{ij}Y_{i'j'}\right] &= \sum_{i,j,i',j'} s_{ij}s_{i'j'}\left(\frac{1}{s(s+1)}\right)^2 \\ &\leq \left(\sum_{i,j} \frac{s_{ij}}{s(s+1)}\right)^2 = \left(\mathbb{E}[Z]\right)^2.\end{aligned}$$

It remains to bound the second term. We know $\mathbb{E}[Y_{ij}Y_{ij'}] \leq 1/s^3$. For a row i of $A_X^T A_X$, let $Z_k(i)$ denote the number of entries in that row which are in level ℓ_k. Then the third sum is

$$\begin{aligned}\mathbb{E}\left[\sum_{i,j,j'} s_{ij}s_{ij'}Y_{ij}Y_{ij'}\right] &\leq s\sum_i \sum_{j,j'} \frac{s_{ij}s_{ij'}}{s^4} \\ &\leq 4s\sum_i \sum_{k,k'} \frac{Z_k(i)Z_{k'}(i)}{s^4 2^{2k}2^{2k'}} \\ &= 4s\sum_i \left(\sum_k \frac{Z_k(i)}{s^2 2^{2k}}\right)^2.\end{aligned}$$

Let $t_i = \sum_k Z_k(i)2^{-2k}$. We now consider the following two cases.

Case 1: At least half of the t_i's are at least s. In this case, we note that

$$\mathbb{E}[Z] \geq \sum_i \sum_k \frac{Z_k(i)}{2^{2k}s^2} = \sum_i \frac{t_i}{s^2} \geq \frac{|X|}{2s} \geq 0.495s.$$

We have

$$4s\sum_i \left(\sum_k \frac{Z_k(i)}{s^2 2^{2k}}\right)^2 \leq 4s\sum_i \sum_k \frac{Z_k(i)}{s^2 2^{2k}} \leq 4s\,\mathbb{E}[Z]$$

where we have used the fact that $\sum_k Z_k(i) \leq s^2$. So altogether,

$$\mathbb{E}[Z^2] \leq 5s\,\mathbb{E}[Z] + \mathbb{E}[Z]^2 \leq 12\,\mathbb{E}[Z]^2.$$

Thus the Paley–Zygmund inequality implies that

$$\Pr(Z \geq 1) \geq (1-0.01)^2 \cdot \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]} \geq \frac{1}{20}.$$

Case 2: At least half of the t_i's are at most s. In this case, let $A_{X'}^TA_{X'}$ be $A_X^TA_X$ restricted to the rows where $t_i \leq s$. We know $|X'| \geq 0.5|X|$. Our goal is to show that $A_{X'}^TA_{X'}$ still has at least 'many' non-zero off-diagonal entries.

The proof is identical to the proof of the three claims $(a), (b), (c)$ above. All sums below only pertain to the matrix $A_{X'}^TA_{X'}$. Indeed, we know $\|A_{X'}\|_F^2 \geq 0.495s^2$ so

$$m \cdot \|A_{X'}^TA_{X'}\|_F^2 = m\sum_i \sigma_i(A_{X'})^4 \geq \left(\sum_i \sigma_i(A_{X'})^2\right)^2 = \|A_{X'}\|_F^4 \geq (.495)^2s^4.$$

Combining with the fact that all non-diagonal entries are bounded by 1.001 in absolute value, the above inequality implies that $\sum_{i\geq 0} \ell_i 2^{-2i+2} \geq (.495)^2s^4/(1.001m) - s^2 \geq 50s^2$. Furthermore, for a row i of $A_{X'}^TA_{X'}$,

$$\sum_k \frac{Z_k(i)}{s^2 2^{2k}} = \frac{t_i}{s^2} \leq \frac{1}{s},$$

so we can bound

$$\begin{aligned} 4s\sum_i \left(\sum_k \frac{Z_k(i)}{s^2 2^{2k}}\right)^2 &\leq 4\sum_i\sum_k \frac{Z_k(i)}{s^2 2^{2k}} \\ &= 4\sum_i \frac{t_i}{s^2} \leq 4\,\mathbb{E}[Z]. \end{aligned}$$

Thus,

$$\mathbb{E}[Z^2] \leq 6\,\mathbb{E}[Z] + \mathbb{E}[Z]^2.$$

So by Payely-Zygmund,

$$\Pr(Z \geq 1) \geq (1 - 0.01)^2 \cdot \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]} \geq \frac{1}{20}.$$

Thus we see that $Z \geq 1$ with probability at least .05 in both cases.

Part #3. Recalling that T is the support set of u, we have $Au \sum_{t\in T} u_tA_t$ and thus,

$$\left\|\sum_{t\in T} u_tA_t\right\|_p^p - \|u\|_p^p = \sum_{i_1,\ldots,i_p} u_{i_1}\cdots u_{i_p}\left(\sum_{j\in[m]} A_{i_1}(j)\cdots A_{i_p}(j)\right) - \sum_{t\in T} u_t^2 := P(u), \tag{4.6}$$

where the outer sum is over all p-sized tuples of indices (with repeats allowed), the notation $A_{i_1}(j)$ denotes the jth entry of the column A_{i_1}, and the inner sum is over all the entries of the column.

We claim that if p is an even integer, then $P(u)$ is a non-zero polynomial. To do so, we demonstrate a monomial $u_{i_1} \cdots u_{i_p}$ with a non-zero coefficient. Now if we assume $Z \geq 1$, then this implies there exists two columns say A_1 and A_2 of A such that $\langle A_1, A_2 \rangle$ is non-zero and both 1 and 2 are in the support set T of u. We consider the following different cases for p.

If $p = 2$, consider the monomial $u_1 u_2$. It's coefficient in P is $\sum_j A_1(j)A_2(j) = \langle A_1, A_2 \rangle \neq 0$, so we are done. Now if $p > 2$ and even, instead consider the monomial $u_1^{p-2} u_2^2$. We claim that this is non-zero. Indeed, since $p - 2$ is also even, $\sum_j A_1(j)^{p-2} A_2(j)^2$ must be non-zero since it is a sum of non-negative terms, one of which is non-zero (since at least one j satisfies $A_1(j)A_2(j) \neq 0$). So we are also done.

Now finally, we note that since the entries of u in its support are picked from a continuous distribution and P is non-zero, the probability of the event $P(u) = 0$ is also 0[3]. However, $\|Au\|_p = \|u\|_p$ implies $\|Au\|_p^p - \|u\|_p^p = P(u) = 0$. Overall, with probability at least 0.05 (the event that $Z \geq 1$), we have $P(u) \neq 0$, contradicting the hypothesis (4.5). Thus, $m \geq s^2/1000$, as desired. ☐

We extend the previous theorem to the case of *approximate* norm preservation for the ℓ_2 case.

Theorem 4.2.1. *Let $A : \mathbb{R}^{s^2} \to \mathbb{R}^m$ be a linear map and $\gamma \leq C/s$ for a sufficiently small constant $C > 0$. If A is such that for any $1 \leq t \leq s$,*

$$\Pr_{x \sim \mathrm{Unif}_t} \left(\left| \|Ax\|_2^2 - \|x\|_2^2 \right| \leq \gamma \|x\|_2^2 \right) \geq 0.99,$$

then $m \geq s^2/1000$.

Proof. Suppose for the sake of contradiction that $m < s^2/1000$. The proof is roughly divided into three parts as in Theorem 4.7.1. The first part shows that $A^T A$ has many 'non-zero' coordinates. The second part shows that a random sparse vector u (as chosen in the hypothesis of the theorem statement) has 'many' pairs of coordinates u_i and u_j such that the corresponding (i, j) entry in $A^T A$ is also non-zero. The last part then shows that the prior result implies that A *does not* (approximately) preserve the norm of u with a large constant probability, contradicting the assumption of the theorem. The last part relies on bounding the probability that a random polynomial lies in an unexpectedly small interval. Since the first two parts are identical, we just present the last part.

Recall the random variable Z defined in the proof of Theorem 4.7.1. There we showed that $Z \geq 1$ with probability at least 0.05.

Part #3. Recalling that T is the support set of u, we have $\|Au\|_2^2 - \|u\|_2^2 = \left\|\sum_{t \in T} u_t A_t \right\|_2^2 - \|u\|_2^2$.

$$\left\| \sum_{t \in T} u_t A_t \right\|_2^2 - \|u\|_2^2 = \sum_{t \neq t' \in T} u_t u_{t'} \langle A_t, A_{t'} \rangle + \sum_{t \in T} \epsilon_t u_t^2 := P(u) \tag{4.7}$$

[3]e.g. see [101]

where $|\epsilon_t| \le 0.001$. If we pick the entries of u to be standard Gaussians, we have

$$\mathbb{E}[P(u)] = \sum_t \epsilon_t.$$

Recalling that Y_{ij} is the indicator for $u_X^T u_X$ having a non-zero entry at (i,j), we have

$$\mathbb{E}[P(u)^2] \ge \sum_{i,j} s_{ij} Y_{ij} + \sum_{t \ne t' \in T} \epsilon_t \epsilon_{t'} + \sum_{t \in T} 3\epsilon_t^2$$

so

$$\mathrm{Var}(P(u)) = \mathbb{E}[P(u)^2] - (\mathbb{E}[P(u)])^2 \ge \sum_{i,j} s_{ij} Y_{ij} = Z.$$

Thus, if we assume that $Z \ge 1$, then Lemma 4.2.2 implies that if the variables of u in the support T are picked from the standard Gaussian distribution, then the probability that $P(u) \in [-\varepsilon, \varepsilon] = O(\sqrt{\varepsilon})$ for any sufficiently small $\varepsilon > 0$. Now consider the following three events.

- $\mathcal{E}_1$ = event that $\|u\|_2^2 \le 100s$.
- $\mathcal{E}_2$ = event that $Z \ge 1$.
- $\mathcal{E}_3$ = event that $P(u) \notin [-c, c]$ for a sufficiently small constant c.

By picking c to be a small enough constant, we know that *all* the events hold with probability at least 0.02. Let's condition on all of these events holding. If so, we show that the condition $|\|Au\|_2^2 - \|u\|_2^2| \le \gamma \|u\|_2^2$ cannot hold. Indeed, the condition directly implies that $\|Au\|_2^2 - \|u\|_2^2 = P(u)$ lies in an interval strictly contained in $[-c, c]$. Thus altogether, with probability at least 0.02, we have $|\|Au\|_2^2 - \|u\|_2^2| > \gamma \|u\|_2^2$, contradicting inequality (4.5). This finishes the proof. $\square$

We now extend the result of the prior theorem to a more general class of mappings. We recall the general class that we consider, as described in Section 4.1. We let $f : \mathbb{R}^{s^2} \to \mathbb{R}^m$ where $f(x) = (f_1(x), \ldots, f_m(x))$ and assume each $f_i : \mathbb{R}^{s^2} \to \mathbb{R}$ is twice differentiable with continuous second partial derivatives.

At a high level, the level of smoothness assumed allows us to consider a taylor expansion, where we approximate each f_i using a linear function up to some quadratic error. By taking the expansion sufficiently close to the origin, the quadratic error becomes negligible, thereby reducing the problem to the linear case. Crucially, we use the fact that our sparse vectors are drawn from a 'scale invariant' distribution, in the sense that scaling a vector sampled from $\mathrm{Unif}_{t,r}$ is equivalent to sampling from $\mathrm{Unif}_{t,r'}$ for an appropriate r'. The full details are given in the proof below.

Theorem 4.7.2. *Suppose $\gamma \le C/s$ for a sufficiently small constant $C > 0$. If $f : \mathbb{R}^{s^2} \to \mathbb{R}^m$ as defined above is such that for any $1 \le t \le s$ and any $r > 0$,*

$$\Pr_{x \sim \mathrm{Unif}_{t,r}} \left(|\|f(x)\|_2^2 - \|x\|_2^2| \le \gamma \|x\|_2^2 \right) \ge 0.999,$$

then $m \ge s^2/1000$.

Proof. Note that we may assume that $f_i(0) = 0$; otherwise our guarantees would trivially fail when $r \to 0$. Since the second partial derivatives of all f_i which comprise f are continuous, they are bounded in magnitude in $[-1,1]^{s^2}$, a compact set. Let L be such an upper bound which holds for all i (note that L may depend on s). Now we let $c \ll 1$ be a sufficiently small value which will be determined shortly. For any fixed f_i, Taylor's theorem for multivariate functions[4] implies that for any $x \in [-c,c]^{s^2}$,

$$|f_i(x) - \langle \nabla f_i(0), x\rangle| \leq L\left(x_1 + \ldots + x_{s^2}\right)^2 \leq Ls^4c^2$$

for every i. Let $A : \mathbb{R}^{s^2} \to \mathbb{R}^m$ be the matrix with a_i as it's rows. The above inequality implies that

$$\|f(x) - Ax\|_\infty \leq c^2 s^4 L.$$

Thus Lemma 4.9.3 implies that

$$|\|f(x)\|_2^2 - \|Ax\|_2^2| \leq c^3 \cdot \mathrm{poly}(s, L) \tag{4.8}$$

for all $x \in [-c,c]^{s^2}$ satisfying $\|f(x)\|_2^2 \leq 2\|x\|_2^2$. By picking c sufficiently small, we can say the following:

- For any $x \in [-c,c]^{s^2}$ such that $|\|f(x)\|_2^2 - \|x\|_2^2| \leq \gamma\|x\|_2^2$, we also have $|\|Ax\|_2^2 - \|x\|_2^2| \leq \gamma\|x\|_2^2 + c^3 \cdot \mathrm{poly}(s,L)$,
- For any x where $|\|Ax\|_2^2 - \|x\|_2^2| \leq \gamma'\|x\|_2^2$ holds for some scalar $\gamma' > 0$, then for any scalar $r > 0$, we also have $|\|Ay\|_2^2 - \|y\|_2^2| \leq \gamma'\|y\|_2^2$ where $y = r \cdot x$.

The first claim follows from inequality (4.8), and the second claim follows from the fact that A is a linear map. Now note that by picking a large enough constant $\beta \gg 1$, we know that sampling from any t, $x \sim \mathrm{Unif}_{t,c/s^\beta}$ satisfies $\|x\|_2^2 \geq c^{2.5}/s^{\beta'}$ (for some other constant $\beta' > 0$) with probability 0.999. Thus a sufficiently small choice of c and large enough β implies the following:

- For any $x \in [-c,c]^{s^2}$ such that $|\|f(x)\|_2^2 - \|x\|_2^2| \leq \gamma\|x\|_2^2$, we also have $|\|Ax\|_2^2 - \|x\|_2^2| \leq 2\gamma\|x\|_2^2$,
- For any t, x sampled from $\mathrm{Unif}_{t,c/s^\beta}$ is in $[-c,c]^{s^2}$ with probability at least 0.999.

Now note that a uniformly chosen vector in $\mathrm{Unif}_{t,r}$ for any $r > 0$ is just a scaled uniformly chosen vector in $\mathrm{Unif}_{t,1}$. Due to our hypothesis on f, it then follows that A is such that for any $1 \leq t \leq s$,

$$\Pr_{x \sim \mathrm{Unif}_{t,1}}\left(\left|\|Ax\|_2^2 - \|x\|_2^2\right| \leq 2\gamma\|x\|_2^2\right) \geq 0.99.$$

The theorem then follows from Theorem 4.2.1 (note that 2γ is *smaller* than the tolerance required in that proof). □

[4] https://en.wikipedia.org/wiki/Taylor's_theorem#Taylor's_theorem_for_multivariate_functions

As discussed in Section 4.1, we further extend our prior result to encoder decoder schemes (recalled below), where another 'decoder' function can be applied on top of the embeddings to compute the ℓ_2 norm. Our lower bound shows that as long as both the encoder and decoder functions are sufficiently smooth, the encoder function is required to map to $\Omega(s^2)$ dimensions. We do not restrict the embedding dimension of the decoder function. In other words, the whole process is 'bottle necked' by the inner dimension.

Definition 4.1.1. *We suppose the encoder and decoder functions satisfy the following.*

- *(Encoder function)* $f : \mathbb{R}^{s^2} \to \mathbb{R}^m$ *where*

$$f(x) = (f_1(x), \dots, f_m(x))$$

  and each $f_i : \mathbb{R}^{s^2} \to \mathbb{R}$ *is twice differentiable with continuous second partial derivatives.*

- *(Decoder function)* $g : \mathbb{R}^m \to \mathbb{R}^{s^2}$ *where*

$$g(x) = (g_1(x), \dots, g_{s^2}(x))$$

  and each $g_i : \mathbb{R}^m \to \mathbb{R}$ *is twice differentiable with continuous second partial derivatives.*

Theorem 4.7.3 (Encoder/Decoder Schemes). *Let* $\gamma \le C/s$ *for a sufficiently small constant* $C > 0$*. If* $h(x) : \mathbb{R}^{s^2} \to \mathbb{R}^{s^2}$ *as defined above is such that for any* $1 \le t \le s$ *and any* $r > 0$*,*

$$\Pr_{x \sim \mathrm{Unif}_{t,r}} \left(|\|h(x)\|_2^2 - \|x\|_2^2| \le \gamma \|x\|_2^2 \right) \ge 0.999,$$

then $m \ge s^2/1000$.

Proof. Similar to the proof of Theorem 4.7.2, let A be the matrix where the ith row is equal to $\nabla h_i(0)$. Note that $A : \mathbb{R}^{s^2} \to \mathbb{R}^{s^2}$. We first claim that A has rank at most m. To see this, note that

$$h(x) = (g_1(f(x)), \dots, g_{s^2}(f(x)).$$

For $1 \le j \le s^2$, let $h_j(x) = g_j(f(x))$. Denoting the input variables of g_j as $g_j(y_1, \dots, y_m)$, the chain rule tells us that for any j and i,

$$\frac{\partial h_j}{\partial x_i} = \sum_{\ell=1}^{m} \frac{\partial g_j}{\partial y_\ell} \cdot \frac{\partial f_\ell}{\partial x_i} = \langle q_i, p_j \rangle$$

where

$$q_i = \left(\frac{\partial f_1}{\partial x_i}, \dots, \frac{\partial f_m}{\partial x_i} \right) \in \mathbb{R}^m$$

and

$$p_j = \left(\frac{\partial g_j}{\partial y_1}, \dots, \frac{\partial g_j}{\partial y_m} \right) \in \mathbb{R}^m.$$

(For simplicity, we are omitting the fact that all the partial derivatives of f_ℓ are being evaluated at 0 and the partial derivatives of g_j are being evaluated $f(0)$). Letting

$$B = \begin{bmatrix} — & q_1^T & — \\ — & q_2^T & — \\ & \vdots & \\ — & q_{s^2}^T & — \end{bmatrix} \in \mathbb{R}^{s^2 \times m}, \quad C = \begin{bmatrix} — & p_1^T & — \\ — & p_2^T & — \\ & \vdots & \\ — & p_{s^2}^T & — \end{bmatrix} \in \mathbb{R}^{s^2 \times m},$$

we see that

$$A = \begin{bmatrix} p_1^T B^T \\ p_2^T B^T \\ \vdots \\ p_{s^2}^T B^T \end{bmatrix} = CB^T \in \mathbb{R}^{s^2 \times s^2}.$$

Thus, A has rank at most m. Now the rest of the proof proceeds by combining elements of Theorem 4.2.1 and 4.7.2, which we only briefly sketch for simplicity.

First, identical to the proof of Theorem 4.7.2, a second-order multi-variable talyor expansion around 0 implies that A is such that for any $1 \le t \le s$

$$\Pr_{x \sim \text{Unif}_{t,1}} \left(\left| \|Ax\|_2^2 - \|x\|_2^2 \right| \le \gamma \|x\|_2^2 \right) \ge 0.99.$$

Now we cannot directly invoke Theorem 4.2.1, since the matrix A in the statement of Theorem 4.2.1 maps $\mathbb{R}^{s^2}$ to $\mathbb{R}^m$, but A is $\mathbb{R}^{s^2} \to \mathbb{R}^{s^2}$. However, note that the proof of Theorem 4.2.1 only relies on the *rank* of the matrix $A^T A$, which is at most m (also true here). Thus, the rest of the proof is identical to the proof of Theorem 4.2.1. □

Finally, we extend the lower bound of Theorem 4.2.1 to the case of approximately preserving inner products.

Theorem 4.7.4. *Let $A : \mathbb{R}^{s^2} \to \mathbb{R}^m$ be a non-zero linear map and $\gamma \le C/s$ for a sufficiently small constant $C > 0$. If A is such that for any $1 \le t, t' \le s$,*

$$\Pr_{x \sim \text{Unif}_t, y \sim \text{Unif}_{t'}} \left(|\langle Ax, Ay \rangle - \langle x, y \rangle| \le \gamma \|x\|_2 \|y\|_2 \right) \ge 0.99$$

and

$$\Pr_{y \sim \text{Unif}_1} \left(\left| \|Ay\|_2^2 - \|y\|_2^2 \right| \le \gamma \|y\|_2^2 \right) \ge 0.99,$$

then $m \ge s^2/1000$.

Proof. The proof is almost identical to that of Theorem 4.2.1 but we present the full details for completeness, since we are working with two vectors instead of one. Suppose for the sake of contradiction that $m < s^2/10^3$. The proof is again roughly divided into three parts. The first part shows that $A^T A$ has many 'non-zero' coordinates. The second part shows that a pair of random sparse vector u and v (as chosen in the hypothesis of the theorem statement) have ‘many’ pairs of coordinates u_i and v_j such that the corresponding (i,j) entry in A^TA is also non-zero. The last part then shows that the prior result implies that A *does not* (approximately) preserve the inner product with a large constant probability, contradicting the assumption of the theorem. The last part relies on bounding the probability that a random polynomial lies in an unusually small interval.

Part #1.

Let $A_1, \ldots, A_{s^2}$ denote the columns of A. Let v be a random vector chosen from Unif_1, and let i be it's support with v_i the corresponding non-zero entry. Then $\|Av\|_2^2 = \|A_i\|_2^2 \cdot v_i^2$ and $\|v\|_2^2 = v_i^2$. Then the hypothesis implies that

$$\Pr(|\|A_i\|_2^2 - 1| \leq 0.0001) \geq 0.99. \tag{4.9}$$

Thus, a 0.99 fraction of the columns of A have Euclidean norm in the range $[0.999, 1.001]$. Let A_X be the restriction to such columns. Note that all diagonal entries of $A_X^TA_X$ are at in the range $[0.999, 1.001]$ and all entries are bounded by 1.001 in absolute value (via Cauchy-Schwarz on the columns of A_X). For $i \geq 1$, we let ℓ_i denote the number of *non-diagonal* entries of $A_X^TA_X$ whose absolute values are in $(2^{-i}, 2^{-i+1}]$ and ℓ_0 denote the rest of the non-diagonal entries (with absolute values in $(1, 1.001]$). We show the following claims hold.

a) $\|A_X\|_F^2 \geq (0.99)^2 s^2$,

b) $\|A_X^TA_X\|_F^2 \geq \frac{(.99)^4 s^4}{m}$,

c) $\sum_{i\geq 0} \ell_i 2^{-2i+2} \geq \frac{(.99)^4 s^4}{m} - 1.001 s^2$.

Claim (a) readily follows from inequality (4.9). To show (b), note that singular values of $A_X^TA_X$ are the squared singular values of A_X so $\|A_X^TA_X\|_F^2 = \sum_{i=1}^m \sigma_i(A_X)^4$ (since the rank of A is at most m). By Cauchy–Schwarz,

$$m \cdot \|A_X^TA_X\|_F^2 = m \sum_i \sigma_i(A_X)^4 \geq \left(\sum_i \sigma_i(A_X)^2\right)^2 = \|A_X\|_F^4 \geq (.99)^4 s^4.$$

To show (c), note that all diagonal entries of $A_X^TA_X$ are at in the range $[0.999, 1.001]$. Claim (c) then follows from using the lower bound of Claim (b) since there are at most s^2 diagonal entries.

Part #2.

Now let u and v be a vector drawn from Unif_s and Unif_r respectively and let T_u and T_v be their corresponding support sets. The hypothesis of the theorem states that

$$\Pr(|\langle Au, Av\rangle - \langle u, v\rangle| \leq \gamma \|u\|_2 \|v\|_2) \geq 0.99. \tag{4.10}$$

Our goal is to demonstrate a contradiction by showing $\Pr(\langle Au, Av\rangle| > \gamma\|u\|_2\|v\|_2) \geq 0.02$, which contradicts (4.10).

Let u_X be the restriction of u to the coordinates in X (i.e., only keep the coordinates in X) and similarly define v_X. Consider $u_X v_X^T$. The non-zero entries of $u_X v_X^T$ lie on a random principal submatrix. We show that with a sufficiently large constant probability, both $u_X v_X^T \in \mathbb{R}^{|X|\times|X|}$ and $A_X^T A_X \in \mathbb{R}^{|X|\times|X|}$ have a non-zero value in 'many' shared entries.

Towards this end, let Y_{ij} be the indicator variable for the entry (i,j) in $u_X u_X^T$ being non-zero and let s_{ij} denote the *squared* (i,j) entry of $A_X^T A_X$. We know

$$\mathbb{E}[Y_{ij}] = \frac{1}{s^2}.$$

Finally, let $Z = \sum_{i,j} s_{ij} Y_{ij}$. Recalling our partitions ℓ_k that we defined earlier and Claim (c), we have

$$\mathbb{E}[Z] = \sum_{i,j} s_{ij}\,\mathbb{E}[Y_{ij}] \geq \frac{1}{s^2}\sum_k \ell_k 2^{-2k} \geq \frac{1}{4s^2}\left(\frac{(.99)^4 s^4}{m} - 1.001 s^2\right) > 200.$$

The same proof as in Theorem 4.2.1 shows that $Z \geq 1$ with probability at least .05.

Part #3.

Recalling that T_u and T_v are the support sets of u and v, we have

$$\langle Au, Av\rangle - \langle u, v\rangle = \sum_{i,j} u_i v_j \langle A_i, A_j\rangle - \sum_i u_i v_i := P(u,v). \tag{4.11}$$

If we pick the entries of u and v to be standard Gaussians, we have $\mathbb{E}[P] = 0$. Recalling that Y_{ij} is the indicator for $u_X^T v_X$ having a non-zero entry at (i,j), we have

$$\mathbb{E}[P^2] \geq \sum_{i,j} s_{ij} Y_{ij} = Z.$$

Thus, if we assume that $Z \geq 1$, then Lemma 4.2.2 implies that if the variables of u and v in their support are picked from the standard Gaussian distribution, then the probability that $P \in [-\varepsilon, \varepsilon] = O(\sqrt{\varepsilon})$ for any sufficiently small $\varepsilon > 0$. Now consider the following three events.

- $\mathcal{E}_1$ = event that $\|u\|_2\|v\|_2 \leq 100s$.
- $\mathcal{E}_2$ = event that $Z \geq 1$.
- $\mathcal{E}_3$ = event that $P \notin [-c, c]$ for a sufficiently small constant c.

By picking c to be a small enough constant, we know that *all* the events hold with probability at least 0.02. Let's condition on all of these events holding. If so, we show that the condition $|\langle Au, Av\rangle - \langle u, v\rangle| \leq \gamma\|u\|_2\|v\|_2$ cannot hold. Indeed, the condition directly implies that $\langle Au, Av\rangle - \langle u, v\rangle = P$ lies in an interval strictly contained in $[-c, c]$. Thus altogether, with probability at least 0.02, we have $|\langle Au, Av\rangle - \langle u, v\rangle| > \gamma\|u\|_2\|v\|_2$, contradicting inequality (4.10). This finishes the proof. $\square$

4.8 Applications

We now present applications of our improved embeddings for sparse vectors for geometric optimization. As stated in the technical overview section, we assume our input is a dataset X of n non-negative s-sparse vectors in $\mathbb{R}^d$. Many of the statements below are also applicable to general sparse vectors (with appropriate modifications), but we focus on non-negative sparse vectors for simplicity.

4.8.1 Diameter

Our goal is to compute the diameter of the dataset X in ℓ_p norm (see Definition 4.1.1).

The first lemma shows that the diameter is preserved when projecting onto very low dimensions.

Lemma 4.8.1. *Let $f : \mathbb{R}^d \to \mathbb{R}^{O(s^2)}$ mapping given in Definition 4.5.1. Let $\tilde{X} = \{f(x) \mid x \in X\} \subset \mathbb{R}^{O(s)}$. We have*

$$\Pr\left(\forall p, \text{diameter}_p(X) = \text{diameter}_p(\tilde{X})\right) \geq 0.99.$$

For the ℓ_∞ case, we can instead embed to $O(s)$ dimensions

Proof. Let (x, y) be a pair in X that witnesses the diameter. Theorem 4.5.1 implies that for any p, the distance between x and y is preserved under f. The proof of the ℓ_∞ case is the same as in Theorem 4.5.2: the coordinate which witnesses the distance between x and y does not collide with any of the other support elements with constant probability. Furthermore, Theorem 4.5.1 also implies that all other distances do not expand under f. This completes the proof. □

This implies the following corollary stating that any 'low dimensional' algorithm for computing the diameter can be used to compute the diameter of X, after composing with our dimensionality reduction.

Corollary 4.8.2. *Given $p \geq 1$, consider an algorithm which computes a C-factor approximation of the diameter of any n point dataset in d-dimensions in $Q(n, d, C)$ time. Then there exists an algorithm which computes a C approximation of the diameter correctly with 99% probability of our dataset X in time $Q(n, O(s^2), C) + O(ns)$. For the ℓ_∞ case, the corresponding bound is $Q(n, O(s), C) + O(ns)$.*

Proof. We simply project our dataset X using f as stated in Lemma 4.8.1 and apply the algorithm Q in the projected space. □

Appealing to existing algorithms on diameter computation implies the following results.

Theorem 4.1.13. *Let X be dataset of non-negative s-sparse vectors. We have the following algorithms:*

1. *Using [52], for any constant integer p, we can compute a $1+\varepsilon$ approximation to* diameter_p *of X in time $\tilde{O}(n/\sqrt{\varepsilon} + 2^{O(s^2 \log(1/\varepsilon))})$ which is correct with probability* 99%.

2. *We can exactly compute the diameter of X in ℓ_∞ norm in time $O(ns)$. This algorithm can be implemented in a stream using $O(s)$ words of memory.*

3. *We can exactly compute the diameter of X in ℓ_1 norm in time $n2^{O(s)}$. This algorithm can be implemented in a stream using $2^{O(s^2)}$ words of memory.*

Proof. The first result just follows from appealing to Corollary 4.1 in [52] which gives an algorithm for computing the diameter in ℓ_p in low-dimensional spaces, and appropriately plugging in the bounds as specified in Corollary 4.8.2.

For the second result, we recall a folklore streaming algorithm for computing diameter in ℓ_∞ norm. We have

$$\text{diameter}_p(X) = \max_{\text{dimension } i} \max_{x,y \in X} |x_i - y_i| = \max_{\text{dimension } i} \left(\max_{x \in X} x_i - \min_{y \in X} y_i \right).$$

Thus, it just suffices to only keep track the maximum and the minimum coordinate along every dimension. Via Lemma 4.8.1, it suffices to assume the dimension is only $O(s)$. The non-streaming algorithm simply scans the points along every dimension as well.

Finally for the third result, we recall a well known isometric embedding of ℓ_1^k into $\ell_\infty^{2^k}$ for any $k \geq 1$: simply map any $x \in \mathbb{R}^d$ to $Ax \in \mathbb{R}^{2^k}$ where A has all possible ± 1 vectors as rows. Applying this embedding reduces the ℓ_1 case to the ℓ_∞ case addressed above. □

4.8.2 Maximum Cut

We consider the max cut problem defined as follows.

Definition 4.8.1. $\text{MaxCut}_p(X) = \max_{S \subseteq X} \sum_{x \in S, y \in X \setminus S} \|x - y\|_p$.

To the best of our knowledge, dimensionality reduction for max cut has been only studied in the ℓ_2 case where its known that $O(1/\varepsilon^2)$ dimensions suffice to estimate the max cut of an arbitrary sized dataset up to $1 \pm \varepsilon$ (only for ℓ_2) [62, 128].

We show the following dimensionality reduction for max cut in the general ℓ_p case.

Theorem 4.8.3. *Let $f : \mathbb{R}^d \to \mathbb{R}^{O(\min(s/\varepsilon^2, s^2/\varepsilon))}$ mapping given in Theorem 4.5.2. Let $\tilde{X} = \{f(x) \mid x \in X\}$. For every $p \geq 1$, we have*

$$\mathbb{E}\left[\left|\text{MaxCut}_p(\tilde{X}) - \text{MaxCut}_p(X)\right|\right] \leq O(\varepsilon) \cdot \text{MaxCut}_p(X).$$

Proof. We drop the dependence on p for the sake of clarity. Let E be the set of edges that participate in a fixed optimal max cut for X. Let $\overline{\text{MaxCut}}(\tilde{X})$ be the value of the cut given by E in the projected dimension (note that E is deterministic but the value is random). We have

$$\begin{aligned}\mathbb{E}\left[\left|\operatorname{MaxCut}(X) - \overline{\operatorname{MaxCut}}(\tilde{X})\right|\right] &\leq \sum_{(x,y)\in E} \mathbb{E}\left[\left|\|f(x)-f(y)\|_p - \|x-y\|_p^p\right|\right] \\ &\leq \sum_{(x,y)\in E} \varepsilon\|x-y\|_p \\ &= \varepsilon \cdot \operatorname{MaxCut}(X),\end{aligned}$$

where we have used the fact that $\|f(x)-f(y)\|_p \geq (1-\varepsilon)\|x-y\|_p$ with probability $1-\varepsilon$ and $0 \leq \|f(x)-f(y)\|_p \leq \|x-y\|_p$ always (refer to the proof of Theorem 4.5.2). Now we make two simple observations. First, we always have

$$\operatorname{MaxCut}(\tilde{X}) \geq \overline{\operatorname{MaxCut}}(\tilde{X}),$$

since $\operatorname{MaxCut}(\tilde{X})$ is the best cut in the projected space. Secondly, since the value of every cut never expands in the projected dimension due to property (3) in Theorem 4.5.1, we also have that the value of *every* cut in the projected space is at most $\operatorname{MaxCut}(X)$ deterministically. In particular, it must also be true that

$$\operatorname{MaxCut}(X) \geq \operatorname{MaxCut}(\tilde{X}).$$

In particular, this means that we have

$$0 \leq \operatorname{MaxCut}(X) - \operatorname{MaxCut}(\tilde{X}) \leq \operatorname{MaxCut}(X) - \overline{\operatorname{MaxCut}}(\tilde{X}),$$

so the random variable $\left|\operatorname{MaxCut}(\tilde{X}) - \operatorname{MaxCut}(X)\right|$ is always bounded by $\left|\operatorname{MaxCut}(X) - \overline{\operatorname{MaxCut}}(\tilde{X})\right|$, which finishes the proof. □

As a corollary, we obtain the following streaming algorithm by combining our dimensionality reduction result above with a known streaming algorithms for max-cut given in [62].

Corollary 4.8.4. *There is a randomized streaming algorithm that, given $0 < \varepsilon < 1/2$, $p \geq 1$, and a non-negative s-sparse $X \subseteq [\operatorname{poly}(n)]^d$ presented as a stream, uses space $\tilde{O}(\operatorname{poly}(\varepsilon^{-1}s))$ and reports an estimate that with probability at least 2/3 is a $(1+\varepsilon)$-approximation to $\operatorname{MaxCut}_p(X)$.*

4.8.3 Clustering Applications

We consider the (arguably) most well-studied formulations of clustering: k-median, k-means, and k-center. Dimensionality reduction for these problems have been well studied in the ℓ_2 case [146]. Here we consider dimensionality reduction for general ℓ_p norms, but with a restricted set of vectors (non-negative sparse).

We recall their definitions below.

Definition 4.8.2. *We are interested in general ℓ_p norm formulations of the k-median/center/means clustering objectives.*

- ***k-median:*** *Given a dataset $X = \{x_1, \dots, x_n\}$ of n points $\in \mathbb{R}^d$, the goal is to find a partition $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ of $[n]$ into k non-empty parts (clusters) to minimize the following:*
$$\text{cost}(\mathcal{C}(X)) = \sum_{i=1}^{k} \min_{u_i \in \mathbb{R}^d} \sum_{j \in C_i} \|x_j - u_i\|_p.$$
- ***k-center:*** *Same as k-median, but we define the cost as*
$$\text{cost}(\mathcal{C}(X)) = \max_{k} \min_{u_i \in \mathbb{R}^d} \max_{j \in C_i} \|x_j - u_i\|_p.$$
- ***k-means:*** *Same as k-median, but we instead use the* squared *distances $\|x - u_i\|_p^2$.*

Note that for these clustering problems, the centers u_i do not have to be in X and can be *arbitrary* points in space. That is, once the points are partitioned, we optimize for the choice of centers. Note that even though our initial dataset may satisfy structural properties as sparsity, is it likely that the optimal centers chosen will not. Thus, while our embedding of Theorem 4.5.2 guarantees that ℓ_p distances between our dataset is preserved, there is no meaningful notion of what the center is under the embedding. Nevertheless, our embedding implies that the cost of *every* clustering is preserved up to a small multiplicative factor.

Theorem 4.8.5. *Consider the k-median or k-center problem. Let X be a set of n non-negative s-sparse vectors in $\mathbb{R}^d$. Let $F : \mathbb{R}^d \to \mathbb{R}^{O(\min(s^2/\varepsilon,\, s/\varepsilon^2)\cdot \log(n)/\varepsilon}$ be the mapping given in Theorem 4.5.2 and $\tilde{X} = \{F(x) \mid x \in X\}$. We have*
$$\Pr\left(\forall \mathcal{C}, \text{cost}(\mathcal{C}(\tilde{X})) = (4 \pm \varepsilon)\text{cost}(\mathcal{C}(X))\right) \geq 1 - 1/\text{poly}(n).$$
For k-means, an identical statement as above holds, except the 4 is replaced by a 16.

Proof. Fix a clustering $\mathcal{C}$ (partition of the datapoints). The complication arises due the fact that $\text{cost}(\mathcal{C}(X))$ optimizes for (non-necessarily sparse) centers in $\mathbb{R}^d$ whereas $\mathcal{C}(\tilde{X}))$ optimizes for centers in the projected space. There may not be an analogue of a center chosen in $\mathbb{R}^d$ in the projected space. To get around this, we show we can simply *move* the centers to their closest data point in X with a small multiplicative loss. The proof proceeds formally as follows.

We first only consider the case of k-median and k-center. Assume that F preserves all pairwise distances between points in X, which holds with probability at least $1 - 1/\text{poly}(n)$ and condition on this event. Take any one of the k partitions C of $\mathcal{C}$ and consider the best center u for this partition in $\mathbb{R}^d$. Now let u' be the closest center to u in its partition C. Moving u to u' will never decrease the cost (by the optimality of u). We claim that it will also never increase the cost by more than a factor of 2. Indeed for k-median, the cost increases by a factor of $|C| \cdot \|u - u'\|_p$ by triangle inequality, which is less than $\sum_{j \in C} \|x_j - u\|_p$ by the minimality of u'. Now the sum is just the original cost. For k-center a similar reasoning holds.

Call such clusterings where the centers are restricted to the dataset points in X as *basic*. The above reasoning implies that given a partition $\mathcal{C}$, the cost of the optimum clusterings and the optimum basic clusterings only differ by a multiplicative factor of 2. Of course the same reasoning is also true in the projected dimension. But note that the costs of *all* basic clusterings are preserved under F, since all pairwise distances are preserved. Thus we have the following chain of inequalities:

$$\text{cost}(\mathcal{C}(\tilde{X}),\ \text{basic}) \le 2\text{cost}(\mathcal{C}(\tilde{X})) \le 2\text{cost}(\mathcal{C}(\tilde{X}),\ \text{basic}) = 2(1 \pm \varepsilon)\text{cost}(\mathcal{C}(X),\ \text{basic}),$$

and similarly

$$2(1 \pm \varepsilon)\text{cost}(\mathcal{C}(X),\ \text{basic}) \le 4(1 \pm \varepsilon)\text{cost}(\mathcal{C}(X)) \le 4(1 \pm \varepsilon)\text{cost}(\mathcal{C}(X),\ \text{basic}).$$

Adjusting ε, altogether, this shows that $\text{cost}(\mathcal{C}(\tilde{X}))$ and $\text{cost}(\mathcal{C}(X))$ are within a factor of $4 \pm \varepsilon$ of each other. The argument does not depend on the choice of $\mathcal{C}$ (once we condition on the event that all pairwise distances are preserved). The proof also easily extends to k-means, where the basic clusterings now increase the cost by at most a factor of 4 due to the squared triangle inequality.

□

4.8.4 Distance Estimation

We first define the distance estimation problem.

Definition 4.8.3. *In the distance estimation problem, we want to preprocess a dataset X and output a datastructure $\mathcal{D}$. Then on any query y, $\mathcal{D}$ outputs an approximation to $\sum_{x \in X} \|x - y\|_p^p$.*

We have the following result.

Theorem 4.8.6. *Given a dataset $X \subset \mathbb{R}^d$ of n non-negative s-sparse vectors and even integer p, we can compute a datastructure $\mathcal{D}$ using $O(n \log(n) ps/\varepsilon)$ preprocessing time. For any fixed non-negative s-sparse query y, $\mathcal{D}(y)$ computes a value t satisfying*

$$\left| t - \sum_{x \in X} \|x - y\|_p^p \right| \le \varepsilon \cdot \sum_{x \in X} \|x - y\|_p^p$$

with probability $1 - 1/\text{poly}(n)$ with query time $O(\log(n) ps/\varepsilon)$.

Proof. We will have $O(\log n)$ independent d-variate polynomials $P_1, \dots P_{O(\log n)}$. Each P_i will output an estimate t_i with the guarantee that

$$\mathbb{E}\left[\left| t_i - \sum_{x \in X} \|x - y\|_p^p \right| \right] \le \varepsilon \cdot \sum_{x \in X} \|x - y\|_p^p,$$

which extends to the high-probability guarantee by just taking medians of all the independent estimates.

P_1 will be constructed by using the 'base' mapping $f_1 : \mathbb{R}^d \to \mathbb{R}^m$ for $m = O(s/\varepsilon)$ of Theorem 4.5.1. We set $\tilde{X}_1 = \{f_1(x) \mid x \in X\}$, and let $P_1(z) = \sum_{x \in \tilde{X}_1} \|x - z\|_p^p$. Then t_1 is simply $P_1(f_1(y))$. Note that each $x \in X$ satisfies $\mathbb{E}[\|f_1(x) - f_1(y)\|_p^p] \geq (1-\varepsilon)\|x-y\|_p^p$ and we always have $\|f_1(x) - f_1(y)\|_p^p \leq \|x-y\|_p^p$, so $0 \leq \mathbb{E}[\|x-y\|_p^p - \|f_1(x) - f_1(y)\|_p^p] \leq \varepsilon\|x-y\|_p^p$. Finally, we repeat this independently for all P_i. The construction and query times are only dependent on the polynomial evaluation.

Note that each P_i is a polynomial with the number of monomials bounded by $O(mp)$. It takes $O(nmp)$ time to construct P and given z, we can compute $P(z)$ in time $O(mp)$ as well. Theorem 4.5.2 guarantees the quality of the approximation.

□

4.9 Auxilliary Lemmas

Lemma 4.9.1. *If $X_1, \ldots, X_t$ are i.i.d. Bernoulli(p) for $p \geq 1 - \varepsilon/100$ then*

$$\Pr\left(\left|\sum_{i=1}^t X_i - tp\right| \geq \varepsilon t\right) \leq \exp\left(-\Omega(\varepsilon t)\right).$$

Proof Sketch. Consider the complement random variables $Y_i = 1 - X_i$. These are Bernoulli$(1-p)$ and so by a standard Chernoff bound,

$$\Pr\left(\left|\sum_{i=1}^t Y_i - t(1-p)\right| \geq \varepsilon t\right) \leq \exp\left(-\Omega(\varepsilon t)\right),$$

as desired. □

Lemma 4.9.2. *Let $z \in \mathbb{R}^d$, $p \geq 10\log(d)/\varepsilon$, and $\varepsilon \in (0, 1/10)$. Then $\|z\|_p \in (1 \pm \varepsilon)\|z\|_\infty$.*

Proof. Without loss of generality, suppose $\|z\|_\infty = |z_1| = \max_{i\in[d]} |z_i|$. We have

$$d|z_1|^p \geq \|z\|_p^p \geq |z_1|^p.$$

Taking $1/p$-th powers gives us

$$d^{1/p}|z_1| \geq \|z\|_p \geq |z_1|,$$

and the lemma follows by noting that $d^{1/p} = \exp(\log(d)/p) = \exp(\varepsilon/10) = 1 + \Theta(\varepsilon)$. □

Lemma 4.9.3. *If $a, b \in \mathbb{R}^d$ satisfy $\|a - b\|_\infty \leq \delta$, then $|\|a\|_2^2 - \|b\|_2^2| \leq 2\delta\sqrt{d}\|a\|_2 + \delta^2 d$.*

Proof. Let $c_i = b_i - a_i$. We know that $|c_i| \leq \delta$. Then $b_i = a_i + c_i$ and $a_i^2 - b_i^2 = a_i^2 - (a_i + c_i)^2 = c_i^2 - 2a_ic_i$ so $|\sum_i (a_i^2 - b_i^2)| \leq d\delta^2 + 2\delta\sum_i |a_i| \leq 2\delta\sqrt{d}\|a\|_2 + \delta^2 d$. □

Chapter 5

Privately Computing Similarities

In this chapter, we give algorithms for computing similarities to private datasets: given a private dataset $X \subset \mathbb{R}^d$ and a similarity function $f(x, y) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, such as a kernel or distance function, output a *private* data structure $\mathcal{D}_X : \mathbb{R}^d \to \mathbb{R}$ which approximates the map $y \to \sum_{x \in X} f(x, y)$. We additionally require that $\mathcal{D}_X$ be always private with respect to X, regardless of the number of times it is queried

In addition to the privacy angle mentioned in Section 1.4, computing similarity to a dataset is a fundamental and well-studied problem in its own right. In the case where f is a kernel such as $f(x, y) = e^{-\|x-y\|_2/\sigma^2}$, this is known as kernel density estimation (KDE), whose non-private setting has been extensively studied [22, 23, 30, 58], with many applications in machine learning; see [98, 173, 174] for a comprehensive overview. In the case where f is a distance function, the sum represents the objective of various clustering formulations, such as k-means and k-median.

5.0.1 Our Results

The aforementioned works have produced non-trivial utility-privacy trade-offs for computing similarities privately for a wide class of f. On the theoretical side, we improve upon these results by giving faster algorithms and improved utility-privacy trade-offs for a wide range of kernels and distance functions. We also study utility lower bounds in order to understand the inherent algorithmic limitations for such problems. Our algorithms are also validated practically; they demonstrate empirical improvements over baselines, both in terms of accuracy and query time.

Definitions. In this work, we consider the natural and standard notion of differential privacy introduced in the seminal work of [76]. Two datasets X, X' are called neighboring if they differ on a single data point.

Definition 5.0.1 ([76])**.** *Let M be a randomized algorithm that maps an input dataset to a range of outputs $\mathcal{O}$. For $\varepsilon, \delta > 0$, M is defined to be (ε, δ)-DP if for every neighboring* *datasets X, X' and every $O \subseteq \mathcal{O}$,*

$$\Pr[M(X) \in O] \leq e^{\varepsilon} \cdot \Pr[M(X') \in O] + \delta.$$

If $\delta = 0$, we say that M is ε-DP, which is referred to as pure differential privacy.

We also work under the function release model; the data structure we output must handle arbitrarily many queries without privacy loss.

Function release. Given a private dataset X and a public function f, we wish to release a differentially private (DP) data structure capable of answering either kernel density estimation (KDE) or distance queries. We focus on the function release model as in [192] and employ their definition: the algorithm designer releases a description of a data structure $\mathcal{D}$ which itself is private (i.e. $\mathcal{D}$ is the output of a private mechanism as per Definition 5.0.1). A client can later use $\mathcal{D}$ to compute $\mathcal{D}(y)$ for any query y. Since $\mathcal{D}$ itself satisfies ε-DP, it can support an *arbitrary number* of queries without privacy loss. This is motivated by scenarios such as synthetic data generation, or when we do not have a pre-specified number of queries known upfront. Our accuracy guarantees are also stated similarly as in [192]: we bound the error for any fixed query y. Thus, while our outputs are always private (since $\mathcal{D}$ itself is private), some query outputs can be inaccurate.

Our private dataset is denoted as $X \subset \mathbb{R}^d$, with $|X| = n$. The similarities are computed with respect to a public function $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. We define distance queries (for distance functions such as $\|x - y\|_2$) and KDE queries (for kernel functions such as $f(x, y) = e^{-\|x-y\|_2^2}$) as follows:

Definition 5.0.2 (Distance Query)**.** *Let f be a distance function. Given a query $y \in \mathbb{R}^d$, a distance query computes an approximation to $\sum_{x \in X} f(x, y)$.*

Definition 1.6.1. (Kernel Density Estimation (KDE) Queries) *For a given dataset $X \subset \mathbb{R}^d$ of size n, kernel function k, and precision parameter $\varepsilon > 0$, a KDE data structure supports the following operation: given a query $y \in \mathbb{R}^d$, return a value $\mathrm{KDE}_X(y)$ that lies in the interval $[(1 - \varepsilon)z, (1 + \varepsilon)\, z]$, where $z = \sum_{x \in X} k(x, y)$, assuming that $k(x, y) \geq \tau$ for all $x \in X$.*

The normalization by $|X| = n$ is inconsequential; we follow prior convention for KDE queries. For distance queries, the un-normalized version seemed more natural to us.

Discussion of Theoretical Results. The main trade-off that we are interested in is between privacy, as measured with respect to the standard DP definition (Definition 5.0.1), and accuracy of our answers, also called utility. For example, a data structure which always returns a fixed answer, such as 42, is clearly always private regardless of the number of queries performed, but is highly inaccurate. Thus, our goal is to obtain non-trivial accuracy guarantees while respecting privacy. Secondary, but important, concerns are query time and data structure construction time and space. Our main theoretical results are summarized in Table 5.1, where we use the following error notation.

Type	f	Thm.	Our Error	Prior Error	Our Query Time	Prior Query Time
Distance Queries	$\|x-y\|_1$	5.3.4	$\left(1+\alpha, \frac{d^{1.5}}{\varepsilon\sqrt{\alpha}}\right)$	$\left(1, \left(\frac{nd^7}{\varepsilon^2}\right)^{1/3}\right)$	d	d [102]
	$\|x-y\|_2$	5.5.1	$(1+\alpha, \frac{1}{\varepsilon\alpha^{1.5}})$	$\left(1, \left(\frac{n^{15}d^7}{\varepsilon^2}\right)^{1/17}\right)$	d	$\geq d$ [102]
	$\|x-y\|_2^2$	5.5.4	$(1, \frac{d}{\varepsilon})$	-	d	-
	$\|x-y\|_p^p$	5.5.3	$\left(1+\alpha, \frac{d}{\varepsilon\sqrt{\alpha}}\right)$	-	d	
KDE Queries	$e^{-\|x-y\|_2}$	5.6.1	$(1, \alpha)$	$(1, \alpha)$	$d+\frac{1}{\alpha^4}$	$\frac{d}{\alpha^2}$ [192]
	$e^{-\|x-y\|_2^2}$	5.6.1	$(1, \alpha)$	$(1, \alpha)$	$d+\frac{1}{\alpha^4}$	$\frac{d}{\alpha^2}$ [192]
	$\frac{1}{1+\|x-y\|_2}$	5.8.1	$(1, \alpha)$	-	$d+\frac{1}{\alpha^4}$	-
	$\frac{1}{1+\|x-y\|_2^2}$	5.8.1	$(1, \alpha)$	$(1, \alpha)$	$d+\frac{1}{\alpha^4}$	$\frac{d}{\alpha^2}$ [192]
	$\frac{1}{1+\|x-y\|_1}$	5.8.1	$(1, \alpha)$	-	$\frac{d}{\alpha^2}$	-

Table 5.1: Summary of the ε-DP upper bounds. See Definition 5.0.3 for the error notation. For clarity, we suppress all logarithmic factors. The KDE bounds assume that $n \geq \tilde{\Omega}\left(\frac{1}{\alpha\varepsilon^2}\right)$. The distance query bounds are stated for points in a bounded radius.

Definition 5.0.3 (Error Notation)**.** *For a fixed query, if Z represents the value output by our private data structure and Z' represents the true value, we say that Z has error (M, A) for $M \geq 1$ and $A \geq 0$ if $\mathbb{E}[|Z - Z'|] \leq (M-1)Z' + A$. That is, we have relative error $M-1$ and additive error A. The expectation is over the randomness used by our data structure.*

We want M to be close to 1 and the additive error A to be as small as possible. Table 5.1 shows our errors and query times, as well as those of the most relevant prior works. See Section 5.1 for a technical overview of how we obtain these results.

For distance queries, the most relevant work is [102]. They considered the ℓ_1 and ℓ_2 functions and obtained additive errors with large dependence on n (dataset size) and d (dimension); see Table 5.1. In contrast, we show that if we allow for a small multiplicative error (e.g. $\alpha = 0.001$ in Table 5.1), we can obtain additive error with improved dependence on d and *no dependence on* n.

Theorem 1.4.1. (Informal; see Theorem 5.3.4 and Corollary 5.5.1) *Suppose the data points have bounded diameter in ℓ_1. For any $\alpha \in (0,1)$ and $\varepsilon > 0$, there exists an algorithm which outputs an ε-DP data structure $\mathcal{D}$ capable of answering any ℓ_1 distance query with α multiplicative error and $\tilde{O}\left(\frac{d^{1.5}}{\varepsilon\sqrt{\alpha}}\right)$ additive error in expectation. For the ℓ_2 case, where the points have bounded ℓ_2 diameter, we instead have $\tilde{O}\left(\frac{1}{\varepsilon\alpha^{1.5}}\right)$ additive error.*

Our approach is fundamentally different, and much simpler, than that of [102], who used powerful black-box online learning results to approximate the sum of distances. Furthermore, given that we think of n as the largest parameter, we incur much smaller additive error. Our simpler approach also demonstrates superior empirical performance as discussed shortly. Our ℓ_1 upper bounds are complemented with a lower bound stating that any ε-DP algorithm supporting ℓ_1 distance queries for private datasets in the box $[0, R]^d$ must incur $\tilde{\Omega}(Rd/\varepsilon)$ error.

Theorem 1.4.2. (Informal; see Theorem 5.4.2) *Any ε-DP data structure which answers ℓ_1 distance queries with additive error at most T for any query must satisfy $T = \tilde{\Omega}(Rd/\varepsilon)$.*

Note that our lower bound only pertains to additive error and does not say anything about multiplicative error. It is an interesting direction to determine if multiplicative factors are also necessary. We also obtain results for other functions related to distances, such as ℓ_p^p; see Section 5.5.

We now discuss our kernel results. For the Gaussian ($e^{-\|x-y\|_2^2}$), exponential ($e^{-\|x-y\|_2}$), and Cauchy ($\frac{1}{1+\|x-y\|_2^2}$) kernels, we parameterize our runtimes in terms of additive error α. Here, we obtain query times of $\tilde{O}(d + 1/\alpha^4)$ whereas prior work [192] requires $\tilde{O}(d/\alpha^2)$ query time. Thus our results are faster in high-dimensional regimes where $d \gg 1/\alpha^2$.

Theorem 5.0.1 (Informal; see Theorems 5.6.1 and 5.8.1)**.** *Consider the Gaussian, exponential, and Cauchy kernels. In each case, for any $\varepsilon > 0$ and $\alpha \in (0, 1)$, there exists an algorithm which outputs an ε-DP data structure that answers KDE queries with error $(1, \alpha)$ and query times $\tilde{O}(d + 1/\alpha^4)$.*

For kernels $\frac{1}{1+\|x-y\|_2}$ and $\frac{1}{1+\|x-y\|_1}$, we obtain the first private data structures; see Table 5.1. We do this via a black-box reduction to other kernels that already have private data structure constructions, using tools from function approximation theory; this is elaborated more in Section 5.1. All KDE results, including prior work, assume that n is lower-bounded by some function of α and ε. These two kernels and the Cauchy kernel fall under the family of *smooth* kernels [22].

We also give faster query times for the *non-private* setting for the Gaussian, exponential, and Cauchy KDEs. Interestingly, our improvements for the non-private setting use tools designed for our private data structures and are faster in the large d regime.

Theorem 5.0.2 (Informal; see Theorem 5.9.1)**.** *For the Gaussian kernel, we improve prior running time for computing a non-private KDE query with additive error α from $\tilde{O}\left(\frac{d}{\varepsilon^2\alpha^{0.173+o(1)}}\right)$ to $\tilde{O}\left(d + \frac{1}{\varepsilon^2\alpha^{2.173+o(1)}}\right)$. Similarly for the exponential kernel, the improvement in the query time is from $\tilde{O}\left(\frac{d}{\varepsilon^2\alpha^{0.1+o(1)}}\right)$ to $\tilde{O}\left(d + \frac{1}{\varepsilon^2\alpha^{2.1+o(1)}}\right)$. The preprocessing time of both algorithms is asymptotically the same as in prior works.*

Discussion of Empirical Results. Our experimental results are given in Section 5.10. We consider three experiments which are representative of our main results. The first setting demonstrates that our ℓ_1 query algorithm is superior to prior state of the art [102] for accurately answering distance queries. The error of our algorithm smoothly decreases as ε increases, but their algorithms always return the trivial estimate of 0. This is due to the fact that the constants used in their theorem are too large to be practically useful. We also demonstrate that our novel dimensionality reduction results can be applied black-box in conjunction with any prior DP-KDE algorithm, leading to savings in both data structure construction time and query time, while introducing negligible additional error.

Lastly, we explore an application to DP classification on the CIFAR-10 dataset. The standard setup is to train a private classification model on the training split (viewed as the private dataset), with the goal of accurately classifying the test split [72, 203]. Our methodology is simple, fast, and does not require a GPU: we simply instantiate a private similarity data structure for each class and assign any query to the class which it has the highest similarity to (or smallest distance if f is a distance). We set f to be ℓ_2^2 since it has arguably the simplest algorithm. In contrast to prior works, our methodology involves no DP-SGD training. For comparable accuracy, we use $>$ **3 orders of magnitude** less runtime compared to prior baselines [72, 203].

5.1 Technical Overview

At a high level, all of our upper bounds crucially exploit fundamental 'low-dimensionality structures' present in the f's that we consider. For different f's, we exploit different 'low-dimensional' properties, elaborated below, which are tailored to the specific f at hand. However, we emphasize that the viewpoint of 'low-dimensionality' is *the* extremely versatile tool driving all of our algorithmic work. We provide the following insights into the low-dimensional properties used in our upper bounds.

Distance Queries via One-dimensional Decompositions. For the ℓ_1 distance function, our improvements are obtained by reducing to the one-dimensional case. To be more precise, we use the well-known property that $\sum_{x \in X} \|x-y\|_1 = \sum_{j=1}^{d} \sum_{x \in X} |x_j - y_j|$. In other words, the sum of ℓ_1 distances decomposes into d one-dimensional sums (this is also true for other related functions such as ℓ_p^p). This explicit low-dimensional representation offers a concrete avenue for algorithm design: we create a differentially private data structure for each of the d one-dimensional projections of the dataset. In one dimension, we can employ many classic and efficient data structure tools. Furthermore, using metric embedding theory (Theorem 2.0.2), we can also embed ℓ_2 into ℓ_1 using an oblivious map, meaning that any algorithmic result for ℓ_1 implies similar results for ℓ_2 as well.

Kernels via New Dimensionality Reduction Results. For kernels such as Gaussian, exponential, and Cauchy, we obtain novel dimensionality reduction results. Our results show that KDE values are preserved if we project both the dataset and the queries to a suitably low dimension via an *oblivious*, data-independent linear map. Our dimensionality reduction schemes are automatically privacy-respecting: releasing an oblivious, data-independent matrix leaks no privacy. Our results also have implications for *non-private* KDE queries and give new state-of-the-art query times.

To obtain our new dimensionality reduction bounds, we analyze Johnson-Lindenstrauss (JL) matrices for preserving sums of kernel values. The main challenge is that kernel functions are *non-linear* functions of distances, and preserving distances (as JL guarantees) does not necessarily imply that non-linear functions of them are preserved. Furthermore, JL-style guarantees may not even be *true*. JL guarantees that distances are preserved up to relative error when projecting to approximately $O(\log n)$ dimensions [119], but this is not possible for the kernel values: if $\|x-y\|_2$ is extremely large, then after applying a JL projection G, $\|Gx-Gy\|_2$ can differ from $\|x-y\|_2$ by a large *additive* factor Δ (even if the relative error is small) with constant probability, and thus $e^{-\|Gx-Gy\|_2} = e^{-\Delta} \cdot e^{-\|x-y\|_2}$ *does not* approximate $e^{-\|x-y\|_2}$ up to relative error.

We overcome these issues in our analysis by noting that we do not require a relative error approximation! Even non-private KDE data structures (such as those in Table 2.1) already incur additive errors. This motivates proving additive error approximation results, where the additive error from dimensionality reduction is comparable to the additive error incurred by existing non-private KDE data structures. We accomplish this via a careful analysis of the non-linear kernel functions and show that it is possible to project onto a *constant* dimension, depending on the additive error, which is independent of the original dataset size n or original dimensionality d.

Theorem 5.1.1 (Informal; see Theorems 5.6.2 and 5.8.2)**.** *Consider the Gaussian and exponential kernels. For any $\alpha \in (0,1)$, projecting the dataset and query to $\tilde{O}(1/\alpha^2)$ dimensions using an oblivious JL map preserves the KDE value up to additive error α. For the Cauchy kernel, projecting to $O(1/\alpha^2)$ dimensions preserves the KDE value up to multiplicative $1+\alpha$ factor.*

We note that variants of 'dimensionality reduction' have been studied for Gaussian kernels, most notably via *coresets* which reduce the dataset size (one can then correspondingly reduce the dimension by projecting onto the data span; see [143, 161]). However, these coresets are data-dependent and it is not clear if they respect DP guarantees. On the other hand, our results use random matrices that do not leak privacy. Lastly, our analysis also sheds additional light on the power of randomized dimensionality reduction *beyond* JL for structured problems, complementing a long line of recent works [33, 39, 56, 68, 114, 146, 156].

Smooth Kernels via Function Approximation Theory. For DP-KDE, we also exploit low-dimensional structures via function approximation, by approximating kernels such as $\frac{1}{1+\|x-y\|_2}$ in terms of exponential functions. To be more precise, for $h(x,y) = \|x-y\|_2, \|x-y\|_2^2$, and $\|x-y\|_1$, Corollary 5.7.2 allows us to express $\frac{1}{1+h(x,y)} \approx \sum_{j\in J} w_j e^{-t_j h(x,y)}$ for explicit parameters t_j, w_j. The corollary follows from a modification of results in approximation theory, see Section 5.7. This can be viewed as projecting the kernel onto a low-dimensional span of exponential functions, since only $|J| = \tilde{O}(1)$ terms in the sum are required. We can then benefit from already existing KDE data structures for various kernels involving exponential functions, such as the exponential kernel! Hence, we obtain new private KDE queries for a host of new functions in a black-box manner. The fact that $|J|$ (the number of terms in the sum) is small is crucial, as instantiating a differentially private KDE data structure for each $j \in [J]$ does not substantially degrade the privacy guarantees or construction and query times. This reduction is detailed in Section 5.7.

5.1.1 Outline of the Chapter

Our ℓ_1 algorithm in one dimension is in Section 5.3. It contains the main ideas for the high-dimensional ℓ_1 algorithm, given in Section 5.3.1. Section 5.4 states our lower bounds for the ℓ_1 distance function. Applications of our ℓ_1 upper bound, such as for ℓ_2 and ℓ_p^p, are given in Section 5.5. Our improved DP bounds for the exponential and Gaussian kernels are given in Section 5.6. Section 5.8 contains improved DP results for smooth kernels (such as Cauchy kernels). Our function approximation theory reduction is presented in Section 5.7. Section 5.9 contains our improved KDE query bounds in the non-private setting. Finally, in Section 5.10, we empirically verify our upper bound algorithms and give applications to private classification.

5.2 Related Work

We use the standard definition of differential privacy [76], given in Definition 5.0.1. We survey the most relevant prior works. We write guarantees in terms of the expected error for any fixed query. These algorithms, and ours, can easily be converted to high-probability results by taking the median of multiple (logarithmic many) independent copies. The theorem statement below pertains to the distance functions $\|x-y\|_1$. It is stated for the case where all the dataset points and queries are in the box $[0,1]^d$, but easily extend to a larger domain by scaling.

Theorem 5.2.1 ([102]). *Assume the dataset and query points are contained in $[0,1]^d$. There exists an algorithm which outputs an ε-DP data structure for the function $\|x-y\|_1$ with the following properties: (1) the expected additive error is $\tilde{O}\left(\frac{n^{1/3}d^{7/3}}{\varepsilon^{2/3}}\right)$, (2) the construction time is $O\left(n^{8/3}\varepsilon^{2/3}d^{2/3}\right)$, (3) the space usage is $O\left(\frac{n^{2/3}\varepsilon^{2/3}}{d^{1/3}}\right)$, (4) and the query time is $O(d)$.*

[102] also obtained results for ℓ_2 with additive errors containing factors of n and d as shown in Table 5.1. The construction times and query times in the result below are not explicitly stated in [102], but they are likely to be similar to that of Theorem 5.2.1.

Theorem 5.2.2 ([102]). *Assume the dataset and query points are contained in the ℓ_2 ball of diameter 1. There exists an algorithm which outputs an ε-DP data structure for the distance function $f(x,y)=\|x-y\|_2$ such that the expected additive error is $\tilde{O}\left(\frac{n^{15/17}d^{7/17}}{\varepsilon^{2/17}}\right)$.*

The result of [192] concerns private KDE constructions for the exponential ($e^{-\|x-y\|_2}$), Gaussian ($e^{-\|x-y\|_2^2}$), and Laplace ($e^{-\|x-y\|_1}$) kernels.

Theorem 5.2.3 ([192]). *Let $\alpha \in (0,1)$ and suppose $n \geq \Omega\left(\frac{1}{\alpha\varepsilon^2}\right)$. For $h(x,y) = \|x-y\|_2, \|x-y\|_2^2$, or $\|x-y\|_1$, there exists an algorithm which outputs an ε-DP data structure for $f(x,y) = e^{-h(x,y)}$ with the following properties: (1) the expected additive error is at most α, (2) the query time is $O\left(\frac{d}{\alpha^2}\right)$, the construction time is $O\left(\frac{nd}{\alpha^2}\right)$, and the space usage is $O\left(\frac{d}{\alpha^2}\right)$.*

Earlier works such as the mechanisms in [7, 38, 91] also study or imply results for DP-KDE. However, many suffer from drawbacks such as exponential dependency on d for running time. The results of [192] were shown to be superior to such prior methods (see therein for more discussions), so we only compare to the current state of the art DP KDE results from [192].

Other Works on Distance Estimation [81] give lower bounds for additive errors of private algorithms which approximate the k-median cost function, which is related to the ℓ_1 distance query. However, their lower bound only applies to coresets specifically, whereas our lower bounds hold for any private mechanism. There have also been recent works designing scalable algorithms for computing distance functions in the non-private setting; see [109] and references therein.

Generic Private Queries. We refer the reader the the excellent survey of [190] and references therein for an overview of algorithms and discussions for broad classes of queries for private datasets. Lastly, we note that dimensionality reduction has been studied in differential privacy in non-KDE contexts in [37, 178]; see the references therein for further related works.

5.3 ℓ_1 Distance Query

We construct a private data structure for answering ℓ_1 distance queries in one dimension. As an overview, the general high-dimensional case, given in Section 5.3.1, can be handled as follows: create a collection of d one-dimensional data structures, constructed on the standard coordinate projections of the dataset. We now describe our one-dimensional algorithm. For the sake of simplicity, let us assume for now that all dataset points are integer multiples of $1/n$ in $[0,1]$. This holds without loss of generality as shown later. Furthermore, let us also instead consider the slightly different *interval* query problem. Here we are given an interval $I \subset \mathbb{R}$ as the query, rather than a point y, and our data structure outputs $|I \cap X|$. We can approximate a distance query at any y by asking appropriate interval queries I, for example geometrically increasing intervals around the query point y.

To motivate the algorithm design, let us additionally ignore privacy constraints for a moment. We use the classic binary tree in one dimension: its leaves correspond to the integer multiples of $1/n$ in $[0,1]$ and store the number of dataset points in that particular position, while internal nodes store the sum of their children. It is well-known that any interval query

I can be answered by adding up the values of only $O(\log n)$ tree nodes. To handle privacy, we release a noisy version of the tree. We note that changing any data point can only change $O(\log n)$ counts in the tree, each by at most one (the leaf to root path). This bounds the sensitivity of the data structure. The formal algorithm and guarantees are stated below. Before presenting them, we make some simplifications which hold without loss of generality.

Remark 5.3.1 (Simplifications)**.** *(1) We scale all dataset points from $[0, R]$ to $[0, 1]$ by dividing by R. We also scale y. We can undo this by multiplying our final estimate by R. (2) After scaling, we assume $y \in [0, 1]$. If y is outside $[0, 1]$, for example if $y \geq 1$, we can just instead query 1 and add $n(y - 1)$ to the final answer, since all dataset points are in $[0, 1]$. This does not affect the approximation. (3) Lastly, we round all points to integer multiples of $1/n$, introducing only $O(R)$ additive error.*

Algorithm 4 Pre-processing data structure

```
1: Input: A set $X$ of $n$ numbers in the interval $[0,1]$, privacy parameter $\varepsilon$
2: Output: An $\varepsilon$-DP data structure
3: procedure Preprocess
4:     Round every dataset point to an integer multiple of $1/n$
5:     Compute the counts of the number of dataset points rounded to every multiple of $1/n$
6:     Build a balanced binary tree $T$ where internal nodes store the sum of the counts of their children and leaf nodes store the counts of the multiples of $1/n$
7:     Independently add noise drawn from Laplace($\eta$) where $\eta = O(\log(n)/\varepsilon)$ to every count
8:     Return tree $T$
9: end procedure
```

Algorithm 5 Interval Query

```
1: Input: Tree $T$, interval $Q \subseteq [0,1]$
2: procedure NoisyCount
3:     Round the endpoints of $Q$ to the closest multiple of $1/n$
4:     Break $Q$ up into the smallest number of contiguous and disjoint pieces such that there is a node in $T$ representing each piece        ▷ At most $O(\log n)$ pieces are required
5:     Return the sum of the counts in each of the nodes in $T$ computed above
6: end procedure
```

Lemma 5.3.1. *The tree T returned by Algorithm 4 is ε-DP.*

Proof. We can encode the tree T as a vector in dimension $O(n)$. Changing one input data point only changes $O(\log n)$ entries of this vector, each by 1, thus the sensitivity of T is $O(\log n)$. Adding coordinate-wise Laplace noise of magnitude $\eta = O(\log(n)/\varepsilon)$ suffices to ensure ε-DP using the standard Laplace mechanism. □

Algorithm 6 One dimensional Distance Query

```
1: Input: data structure $T$ from Algorithm 4, query $y \in [0,1]$, accuracy parameter $\alpha \in (0,1)$.
2: procedure DistanceQuery
3:     Round $y$ to the closest integer multiple of $1/n$
4:     Value $\leftarrow 0$
5:     for $j = 0, 1, ..., O(\log(n)/\alpha)$ do
6:         $Q_j \leftarrow \left[y + \frac{1}{(1+\alpha)^{j+1}}, y + \frac{1}{(1+\alpha)^j}\right)$    ▷ This will consider the points to the right of $y$
7:         Value $\leftarrow$ Value + NoisyCount$(Q_j) \cdot \frac{1}{(1+\alpha)^j}$
8:     end for
9:     Repeat the previous loop for intervals to the left of $y$
10:    Return Value
11: end procedure
```

We now analyze the utility of the algorithm.

Theorem 5.3.2. *Suppose $X \subseteq [0,R]$ is a dataset of n numbers in one dimension. Let $\alpha \in (0,1)$ be the accuracy parameter used in Algorithm 6. Let A be the output of Algorithm 6 and let $A' = \sum_{x \in X} |y - x|$ be the true distance query value. Then we have $\mathbb{E}\,|A - A'| \le \alpha A' + \tilde{O}\left(\frac{R}{\varepsilon\sqrt{\alpha}}\right)$.*

Proof. For simplicity, we only consider the distance query to the points in X to the right of y. The identical proof extends to the symmetric left case. We also work under the simplifications stated in Remark 5.3.1. They only affect the additive error by at most $O(R)$. For an interval Q, define TrueCount(Q) to be the true value $|Q \cap X|$. Let

$$\text{Estimate}_1 = \sum_{j \ge 0} \frac{1}{(1+\alpha)^j} \cdot \text{TrueCount}(Q_j) \qquad \text{and} \qquad A' = \sum_{j \ge 0} \sum_{x \in X \cap Q_j} |y - x|.$$

First, we know that $|\text{Estimate}_1 - A'| \le \alpha \cdot A'$, as for all j, the distanced between y and different points $x \in X \cap Q_j$ only differ by a multiplicative $(1+\alpha)$ factor. Thus, it suffices to show the output as returned by Algorithm 6, i.e. A, differs from Estimate$_1$ by $\tilde{O}\left(\frac{1}{\varepsilon\sqrt{\alpha}}\right)$. Let NoisyCount$(Q)$ denote the interval query answer returned by our noised-tree via Algorithm 5. Algorithm 6 outputs $A = \sum_{j \ge 0} \frac{1}{(1+\alpha)^j} \cdot \text{NoisyCount}(Q_j)$. We wish to bound

$$|\text{Estimate}_1 - A| \le \left| \sum_{j \ge 0} \frac{1}{(1+\alpha)^j} \cdot (\text{TrueCount}(Q_j) - \text{NoisyCount}(Q_j)) \right|.$$

Note that $Z_j := \text{TrueCount}(Q_j) - \text{NoisyCount}(Q_j)$ is equal to the sum of at most $O(\log n)$ Laplace random variables, each with parameter $O((\log n)/\varepsilon)$. This is because we compute all noisy counts by accumulating the counts stored in the individual nodes in T corresponding to Q_j. We only query $O(\log n)$ nodes for any Q_j and each node has independent noise added. Thus, $\mathbb{E}\, Z_j = 0$ and $\mathrm{Var}\left[Z_j \cdot \frac{1}{(1+\alpha)^j}\right] \le \frac{\tilde{O}(1)}{\varepsilon^2} \cdot \frac{1}{(1+\alpha)^{2j}}$. In addition, the Z_j's are also independent of each other since the intervals Q_j's are disjoint, meaning we query disjoint sets of nodes in the tree for different Q_j's. Hence,

$$\mathrm{Var}\left[\sum_{j \ge 0} \frac{1}{(1+\alpha)^j} \cdot Z_j\right] \le \frac{\tilde{O}(1)}{\varepsilon^2} \cdot \sum_{j \ge 0} \frac{1}{(1+\alpha)^{2j}} \le \frac{\tilde{O}(1)}{\alpha \varepsilon^2}, \tag{5.1}$$

meaning with large constant probability, say at least 0.999, the quantity $|\text{Estimate}_1 - A|$ is at most $\tilde{O}(1)/(\varepsilon\sqrt{\alpha})$ by Chebyshev's inequality. A similar conclusion also holds in expectation since for any centered random variable W, $\mathbb{E}\,|W| \le \sqrt{\mathrm{Var}(W)}$. We recover our desired statement by multiplying through by R to undo the scaling. □

5.3.1 High-Dimensional ℓ_1 Query

Algorithm 6 automatically extends to the high dimensional case due to the decomposability of the ℓ_1 distance function. Indeed, we simply instantiate d different one-dimensional distance query data structures, each on the coordinate projection of our private dataset. The algorithm is stated below. For simplicity, we state both the preprocessing and query algorithms together.

Algorithm 7 High-dimensional ℓ_1 distance query

```
1: Input: Set $X$ of $n$ $d$-dimensional points in the box $[0, R]^d$, privacy parameter $\varepsilon$, multiplicative accuracy parameter $\alpha$, query $y$
2: procedure $\ell_1^d$ Query
3:     Instantiate $d$ different one-d data structures $\mathcal{D}_1, \ldots, \mathcal{D}_d$. $\mathcal{D}_i$ is the output of Algorithm 4 on the $i$th coordinate projections of $X$. Each data structure is $\varepsilon/d$-DP ▷ Preprocessing Stage
4:     Return The sum of outputs when $\mathcal{D}_i$ is queried on $y_i$ for every $i$    ▷ Query Stage
5: end procedure
```

Our result is stated in Theorem 5.3.4, which is a corollary of Lemma 5.3.1, Theorem 5.3.2, and the following advanced composition theorem of [77].

Theorem 5.3.3 (Advanced Composition Starting from Pure DP [77])**.** *Let $M_1, \ldots, M_k : \mathcal{X}^n \to \mathcal{Y}$ be randomized algorithms, each of which is ε-DP. Define $M : \mathcal{X}^n \to \mathcal{Y}^k$ by $M(x) = (M_1(x), \ldots, M_k(x))$ where each algorithm is run independently. Then M is (ε', δ)-DP for any $\varepsilon, \delta > 0$ and*

$$\varepsilon' = \frac{k\varepsilon^2}{2} + \varepsilon\sqrt{2k\log(1/\delta)}.$$

For $\delta = 0$, M is $k\varepsilon$-DP.

Theorem 5.3.4. *Let A be the output of Algorithm 7. Let $A' = \sum_{x \in X} \|y - x\|_1$ be the true answer. We have $\mathbb{E}\,|A - A'| \le \alpha A' + \tilde{O}\left(\frac{Rd^{1.5}}{\varepsilon\sqrt{\alpha}}\right)$. Furthermore, Algorithm 7 is ε-DP.*

Proof. The ε-DP guarantee follows from standard DP composition results (Theorem 5.3.3), so it remains to argue about the approximation guarantee. Let A_i be the estimate returned by $\mathcal{D}_i$ and let A'_i be the true answer in the ith dimension. Note that $A' = \sum_i A'_i$ and $A = \sum_i A_i$. Naively applying Theorem 5.3.2 gives us additive error $\tilde{O}(Rd^2/(\varepsilon\sqrt{\alpha}))$. However, we can exploit the fact that the data structures in the individual dimensions are using independent randomness to get a better bound.

Let us inspect the proof of Theorem 5.3.2. Let Z_j^i be the variables Z_j used in the proof of Theorem 5.3.2 for coordinate i. Similar to the proof of Theorem 5.3.2, we can note that the error incurred by our estimate among *all* coordinates, can be upper bounded by the absolute value of $\sum_j \frac{1}{(1+\alpha)^j}\left(\sum_i Z_j^i\right)$, where each Z_j^i are independent across i and j and are each the sum of at most $O(\log n)$ different Laplace $O((\log n)/\varepsilon)$ random variables. The variance of each individual dimension is given by Equation 5.1 (with ε scaled down by ε/d), i.e., it is of the order $\tilde{O}(d^2)/(\alpha\varepsilon^2)$. The total variance across d copies is then $\tilde{O}(d^3)/(\alpha\varepsilon^2)$. Finally, the same calculations as the proof of Theorem 5.3.2 imply an additive error of the square root of this quantity, namely $\tilde{O}(d^{1.5})/(\sqrt{\alpha}\varepsilon)$. □

If we relax the privacy guarantees to approximate DP, we can get a better additive error, matching the additive error term in the lower bound of Theorem 5.4.2 (note however that Theorem 5.4.2 is a lower bound on pure DP algorithms, not approximate DP).

Theorem 5.3.5. *Let $\delta > 0$ and A be the output of Algorithm 7 where every one dimensional algorithm is instantiated to be $(c\varepsilon/\sqrt{d\log(1/\delta)})$-DP for a sufficiently small constant c independent of all parameters. Let $A' = \sum_{x \in X} \|y - x\|_1$ be the true answer. We have $\mathbb{E}\,|A - A'| \le \alpha A' + \tilde{O}\left(\frac{Rd\sqrt{\log(1/\delta)}}{\varepsilon\sqrt{\alpha}}\right)$. Furthermore, Algorithm 7 is (ε, δ)-DP assuming $\varepsilon \le O(\log(1/\delta))$. $\tilde{O}$ hides logarithmic factors in n.*

Proof. The same proof as Theorem 5.3.4 applies, but instead we use the approximate DP advanced composition result 5.3.3, and an appropriately smaller noise parameter in Algorithm 4. □

5.4 Lower Bounds for ℓ_1Distance Queries

First we obtain lower bounds for the one-dimensional case, which is then extended to the arbitrary dimensional case. Recall the definition of the dimensional distance query problem: Given a dataset X of n points in the interval $[0, R]$, an algorithm outputs a data structure which given query $y \in \mathbb{R}$, computes the value $\sum_{x \in X} |x - y|$. Our lower bound idea is via the 'packing lower bound' technique used in DP [34, 96, 191]. At a high level, we construct many datasets which differ on very few points. By the restrictions of DP, the ℓ_1 distance queries on these datasets must be similar, since they are all ‘nearby’ datasets. However, our construction will ensure that these different datasets result in vastly different *true* ℓ_1 distance queries for a fixed set of queries. This implies a lower bound on the additive error incurred from the privacy requirements.

Theorem 5.4.1. *For sufficiently large n and any $\varepsilon < 0.2$, any ε-DP algorithm which outputs a data structure such that with probability at least $2/3$, the distance query problem is correctly answered on any query y with additive error at most T, must satisfy $T = \Omega(R/\varepsilon)$.*

Proof. Let T be the additive error of an algorithm $\mathcal{A}$ as described in the theorem statement. Our goal is to show that we must have $T \geq \Omega(R/\varepsilon)$. Note that crucially we know the value of T. Since the purported algorithm outputs an ε-DP data structure, we use the value of T to design a ‘hard’ instance for the algorithm.

Let $\alpha \in [0, 1]$ be a parameter. Define $2^{R^{1-\alpha}}$ different datasets as follows: we first put markers in the interval $[0, R]$ at locations kR^α for all $0 \leq k \leq R^{1-\alpha}$. At every marker besides 0, we either put 0 or $\gamma := \lceil \frac{3T}{R^\alpha} \rceil$ data points, and consider all such possible choices. The rest of the data points are put at location 0. For sufficiently large n, this results in the claimed number of datasets.

Now for every such dataset D, define the function $f_D : \mathbb{R} \to \mathbb{R}$ defined as

$$f_D(y) = \sum_{x \in D} |x - y|.$$

We claim that the (exact) vector of evaluations

$$[f_D(0), f_D(R^\alpha), f_D(2R^\alpha), \ldots, f_D(R), f_D(R + R^\alpha)]$$

uniquely determines D. Indeed, f_D is a piece wise linear function consisting of at most $R^{1-\alpha} + 2$ pieces. Its slopes can only change precisely at the locations kR^α. Thus, exactly calculating $f_D((k+1)R^\alpha) - f_D(kR^\alpha)$ gives us the exact values of the slopes, and thus allows us to reconstruct the piece wise linear functions that comprise f_D. Correspondingly, this allows us to determine which markers contain a non zero (i.e. γ) number of points, reconstructing D.

The second claim is that the vector of evaluations with entry wise additive error at most T allows for the *exact* reconstruction of the vector of evaluations. This follows from the fact that the exact evaluation values are multiples of γR^α and the additive error is small enough to determine the correct multiple. Formally, we have that $T < \frac{1}{2}\gamma R^\alpha$ and since each $f_D(kR^\alpha)$ is a multiple of γR^α, any entry of a noisy evaluation vector with additive error at most T can be easily rounded to the correct value, as it lies closest to a *unique* multiple of γR^α.

Now the rest of the proof proceeds via the ‘packing’ argument for proving lower bounds in differential privacy. Let Q be the queries defined above and let P_D be the set of allowable vectors of evaluations (i.e. those that achieve entry wise error of at most T) for dataset D on Q. As argued above, the probability that $\mathcal{A}$ on dataset D outputs a vector in P_D is at least $2/3$, and all these sets P_D are disjoint as argued above. Furthermore, all datasets differ in at most

$$\gamma R^{1-\alpha} \le 3TR^{1-2\alpha} + R^{1-\alpha}$$

data points. Let D' be the dataset with all points at 0. Group privacy gives us

$$\begin{aligned}
1 &\ge \sum_D \Pr(\mathcal{A}(D', Q) \in P_D) \\
&\ge \sum_D e^{-(3TR^{1-2\alpha}+R^{1-\alpha})\varepsilon} \cdot \Pr(\mathcal{A}(D, Q) \in P_D) \\
&\ge \sum_D \frac{2}{3} e^{-(3TR^{1-2\alpha}+R^{1-\alpha})\varepsilon} \\
&\ge 2^{R^{1-\alpha}} \cdot \frac{2}{3} e^{-(3TR^{1-2\alpha}+R^{1-\alpha})\varepsilon}.
\end{aligned}$$

It follows that

$$3TR^{1-2\alpha} + R^{1-\alpha} \ge \frac{\log(2)R^{1-\alpha}}{\varepsilon} + \frac{\log(2/3)}{\varepsilon} \implies T \ge \frac{\log(2)R^{\alpha}}{3\varepsilon} + \frac{\log(2/3)}{3\varepsilon R^{1-2\alpha}} - \frac{R^{\alpha}}{3}.$$

Taking $\alpha \to 1$, we can check that for $\varepsilon \le 0.2$, $T \ge 0.02R/\varepsilon = \Omega(R/\varepsilon)$, as desired. $\square$

Let us now extend the lower bound to d dimensions. Recall the problem we are interested in is the following: Given a dataset X of n points in $\mathbb{R}^d$ with every coordinate in the interval $[0, R]$, give an algorithm which outputs a data structure, which given a query $y \in \mathbb{R}^d$, computes the value $\sum_{x \in X} \|x - y\|_1$. The goal is to prove that $\approx Rd/\varepsilon$ additive error is required for answering queries of this form. For a vector v, let $v(j)$ denote its jth coordinate.

Theorem 5.4.2. *For sufficiently large n and R as a function of d and sufficiently small constant ε, any ε-DP algorithm which outputs a data structure which with probability at least $2/3$ answers the above query problem for any query with additive error at most T, must satisfy $T = \tilde{\Omega}(Rd/\varepsilon)$.*

Proof. We reduce the 1 dimensional version of the problem to the d dimensional version which allows us to use the lower bound of Theorem 5.4.1. Pick α such that $(Rd)^\alpha = Rd/\log(Rd)$ and suppose R satisfies $R \ge 2^{Cd}$ for a sufficiently large constant C. Furthermore, assume that R is an integer multiple of R^α. Now consider the lower bound construction from Theorem 5.4.1 where the parameter 'R' there is replaced by Rd and α is as stated. Theorem 5.4.1 implies that an ε-DP data structure which with probability at least $2/3$ correctly answers any distance query for a *one dimensional* input y must have additive error at least $\Omega((Rd)^\alpha/\varepsilon) = \tilde{\Omega}(Rd/\varepsilon)$. We will now show how to simulate any one dimensional query y on this lower bound instance with *one* d dimensional query on a related d dimensional instance.

To construct the d dimensional instance, consider the interval $[0, Rd]$ as d different blocks, separated by integer multiples of R as $[0, R), [R, 2R), \ldots$ etc. Note that in the one dimensional hard instance we are considering from Theorem 5.4.1, we can always ensure that every one of these d blocks contains the same number of points (For example by only considering such 'balanced' allocations of dataset constructions in the marker construction from 5.4.1. Due to our choice of R and α, it is easy to see that the number of such balanced allocations is at least $2^{\Theta((Rd)^{1-\alpha})}$). Let X_1 be this one dimensional dataset and let n' be the number of (one dimensional) points that are contained within each block. Consider the d dimensional dataset on n' points where the (one dimensional) points in the first block are the first coordinate projections of the dataset and in general, the points in the ith block are the ith coordinate projections of the dataset. Since every block has the same number of points, we can construct such a dataset which is consistent with respect to these coordinate projections[1]. Denote this dataset by X_d. For a one dimensional query y, make a vector $\hat{y} \in \mathbb{R}^d$ which just has y copied in all its coordinates.

We have

$$\begin{aligned}\sum_{x \in X_1} |y - x| &= \sum_{\text{blocks } b} \sum_{x \in b} |y - x| \\ &= \sum_{j=1}^{d} \sum_{x \in X_d} |x(j) - \hat{y}(j)| \\ &= \sum_{x \in X_d} \|x - \hat{y}\|_1.\end{aligned}$$

Thus, the exact value of the single d dimensional query we have constructed is equal to the exact value of the one dimensional query we are interested in. This reduction immediately implies that any ε-DP data structure which with probability at least 2/3 answers all d dimensional queries with additive error at most T must satisfy $T = \tilde{\Omega}(Rd/\varepsilon)$, as desired. □

Remark 5.4.1. *The lower bound is slightly stronger than stated since we only assume the query vectors have their one dimensional coordinates bounded by $\tilde{O}(R)$.*

Remark 5.4.2. *There is a gap between our upper and lower bounds. Our ε-DP data structure has a $O(d^{1.5})$ dependency whereas the lower bound we prove only states $\Omega(d)$ dependency is required. Note that our approx-DP result of Theorem 5.3.5 has only $O(d)$ dependency, but the lower bound we proved only applies to pure DP algorithms. It is an interesting question to close this gap between the upper and lower bounds.*

5.5 Corollaries of Our ℓ_1 data structure

Our high dimensional ℓ_1 distance query result implies a multitude of downstream results for other distances and functions. For example, it automatically implies a similar result for the ℓ_2 case via a standard oblivious mapping from ℓ_2 to ℓ_1. A similar procedure was applied in

[1]Note there are many ways to assign the coordinate projections to the points in $\mathbb{R}^d$. We can just consider one fixed assignment.

[102], but using our version of the ℓ_1 distance query obtains superior results for the ℓ_2 case. The guarantees of the mapping is stated in Theorem 2.0.2.

The mapping T given by Theorem 2.0.2 is *oblivious* to the private dataset and can be released for free without any loss in privacy, and the distances are preserved up to a $1+\alpha$ multiplicative error if we take $k = O(\log(n)\log(1/\alpha)/\alpha^2)$ for a dataset of size n.

Corollary 5.5.1. *Let X be a private dataset of size n with a bounded diameter of R in ℓ_2. There exists an ε-DP data structure such that for any fixed query y, with probability 99%, it outputs Z satisfying $\left|Z - \sum_{x\in X}\|x-y\|_2\right| \le \alpha\sum_{x\in X}\|x-y\|_2 + \tilde{O}\left(\frac{R}{\alpha^{1.5}\varepsilon}\right)$.*

Proof. We sketch the argument since it is follows from a combination of the prior listed results. We just apply the embedding result of Theorem 2.0.2 as well as the guarantees of our ℓ_1 distance query data structure from Theorem 5.3.4. The only thing left to check is the bounds of the individual coordinates after we apply the embedding. Note that with high probability, every coordinate after applying T will be bounded by $\tilde{O}(R\alpha^2)$.[2] The bound follows by just plugging into the statement of Theorem 5.3.4. □

Note that minor modifications to the one dimensional ℓ_1 algorithm also implies the following:

Corollary 5.5.2. *Let $p \ge 1$ and suppose all points in our one-dimensional datasets are in the interval $[0, R]$. Let Z be the output of Algorithm 6 but with α scaled down by a factor of p and all counts weighted instead by $(R/(1+\alpha/p)^j)^p$ in Lines 8 and 13 of Algorithm 6. Let $Z' = \sum_{x\in X}|y-x|^p$ be the true answer. We have $\mathbb{E}|Z - Z'| \le \alpha Z' + \tilde{O}\left(\frac{R^p\log(1/p)}{\varepsilon\sqrt{\alpha}}\right)$.*

The higher dimensional version also follows in a similar manner to Theorem 5.3.4 due to the decomposability of $\ell_p^p : \sum_{x\in X}\|x-y\|_p^p = \sum_{j=1}^d\sum_{x\in X}|x(j)-y(j)|^p$.

Corollary 5.5.3. *Let Z be the output of Algorithm 7 but with modifications made as in Corollary 5.5.2. Let $Z' = \sum_{x\in X}\|y-x\|_p^p$ be the true answer. We have, $\mathbb{E}|Z-Z'| \le \alpha Z' + \tilde{O}\left(\frac{R^p d^{1.5}\log(1/p)}{\varepsilon\sqrt{\alpha}}\right)$. Furthermore, Algorithm 7 is ε-DP. Similarly to Theorem 5.3.5, we can get an (ε,δ)-DP algorithm satisfying $\mathbb{E}|Z-Z'| \le \alpha Z' + \tilde{O}\left(\frac{R^p d\sqrt{\log(p/\delta)}}{\varepsilon\sqrt{\alpha}}\right)$.*

5.5.1 An alternate algorithm for ℓ_2^2

We give an alternate, simpler algorithm, with slightly better guarantees than our general ℓ_p^p result.

Corollary 5.5.4. *There exists an ε-DP algorithm which answers the ℓ_2^2 distance query with additive error $O\left(\frac{R^2 d}{\varepsilon}\right)$ in expectation and requires $O(d)$ query time.*

[2]We need to clip the the coordinates of the embedding output so that every coordinate lies in the correct range. But this event happens with high probability, so it does not affect the distance-preserving guarantees.

Proof. The following identity holds:

$$\sum_{x\in X} \|x-y\|_2^2 = \sum_{x\in X} \|x-\mathbb{E}\,X\|_2^2 + n\|y-\mathbb{E}\,X\|_2^2$$

where $\mathbb{E}\,X = \frac{1}{n}\sum_{x\in X} x$. Note that the first quantity $\sum_{x\in X}\|x-\mathbb{E}\,X\|_2^2$ is a scalar which does not depend on the query y. Thus, an alternate ε-DP algorithm in the ℓ_2^2 case is to first release a (noisy) version of $\sum_{x\in X}\|x-\mathbb{E}\,X\|_2^2$ as well as a noisy $\mathbb{E}\,X$.

If all coordinates are in $[0,R]$, then changing one data point can change every coordinate of $\mathbb{E}\,X$ by a R/n factor. Analyzing $\sum_{x\in X}\|x-\mathbb{E}\,X\|_2^2$ is a bit trickier since changing one data point changes a term in the sum as well as $\mathbb{E}\,X$. Let z denote the new mean after changing one data point in X and let $\mathbb{E}\,X$ denote the old mean. We have

$$\begin{aligned}\sum_{x\in X}\|x-z\|_2^2 &= \sum_{x\in X}\|x-\mathbb{E}\,X+\mathbb{E}\,X-z\|_2^2\\ &= \sum_{x\in X}\left(\|x-\mathbb{E}\,X\|_2^2 + 2\langle x-\mathbb{E}\,X, \mathbb{E}\,X-z\rangle + \|\mathbb{E}\,X-z\|_2^2\right).\end{aligned}$$

Now $n\|\mathbb{E}\,X-z\|_2^2 \le O(R^2d/n)$ and

$$\sum_{x\in X} 2|\langle x-\mathbb{E}\,X,\mathbb{E}\,X-z\rangle| \le 2\sum_{x\in X}\|x-\mathbb{E}\,X\|_2\cdot\|z-\mathbb{E}\,X\|_2 \le O(R^2d).$$

Thus, the simple Laplacian mechanism of adding Laplace($O(R^2d/\varepsilon)$) and releasing the value of $\sum_{x\in X}\|x-\mathbb{E}\,X\|_2^2$ ensures $\varepsilon/2$-DP. Then we can release the vector $\mathbb{E}\,X$ by adding Laplace($O(Rd/(n\varepsilon))$) noise to every coordinate, to also ensure $\varepsilon/2$-DP. Overall, the algorithm is ε-DP. To analyze the error, note that we get additive error $O(R^2d/\varepsilon)$ from the noisy value $\sum_{x\in X}\|x-\mathbb{E}\,X\|_2^2$. Assuming n is sufficiently large, we can easily repeat a calculation similar to above which shows that the overall additive error is at most $O(R^2d/\varepsilon)$ in expectation. Indeed, letting z denote the noisy mean we output, we have

$$\left|\|y-z\|_2^2 - \|y-\mathbb{E}\,X\|_2^2\right| \le 2\|y-z\|_2\cdot\|z-\mathbb{E}\,X\|_2 + \|z-\mathbb{E}\,X\|_2^2,$$

from which the conclusion follows. □

5.6 Improved Bounds for the Exponential and Gaussian Kernels

In this section we provide our bounds for the exponential and Gaussian kernels, improving the query time of the result of [192] stated in Theorem 5.2.3.

Theorem 5.6.1. *Let $\alpha \in (0,1)$ and suppose $n \ge O\left(\frac{1}{\alpha\varepsilon^2}\right)$. For $h(x,y) = \|x-y\|_2$ and $\|x-y\|_2^2$, there exists an algorithm which outputs an ε-DP data structure for the kernel $f(x,y) = e^{-h(x,y)}$ with the following properties:*

1. The expected additive error is α,

2. The query time is $\tilde{O}\left(d + \frac{1}{\alpha^4}\right)$,

3. The construction time is $\tilde{O}\left(nd + \frac{n}{\alpha^4}\right)$,

4. and the space usage is $\tilde{O}\left(d + \frac{1}{\alpha^4}\right)$.

Note that our bound improves upon [192] in the large dimension regime $d \gg 1/\alpha^2$, by disentangling the factors of d and $1/\alpha$. We prove this via a general dimensionality reduction result, which maybe of general interest. Note that our dimensionality reduction result also implies improved bounds for KDE queries in the non-private setting as well, as elaborated in Section 5.9.

5.6.1 Dimensionality Reduction for Gaussian KDE

We obtain general dimensionality reduction results for the Gaussian and exponential KDE, using variants of the Johnson-Lindenstrauss (JL) transforms. See 5.1 for an overview and motivations.

Theorem 5.6.2 (Dim. Reduction for Gaussian and exponential kernels)**.** *Let $G : \mathbb{R}^d \to \mathbb{R}^{O(\log(1/\alpha)/\alpha^2)}$ be the standard Gaussian JL projection where $\alpha < 1$ is a sufficiently small constant. Fix a query $y \in \mathbb{R}^d$. Let*

$$z = \frac{1}{|X|} \sum_{x \in X} f(x, y),$$
$$\hat{z} = \frac{1}{|X|} \sum_{x \in X} f(Gx, Gy)$$

for $f(x, y) = e^{-\|x-y\|_2}$ or $f(x, y) = e^{-\|x-y\|_2^2}$. Then, $\mathbb{E}\,|z - \hat{z}| \le \alpha$.

As stated, Theorem 5.6.2 requires a projection matrix of dense Gaussian random variables, making the projection time $\tilde{O}(d/\alpha^2)$. We can speed this up by using the *fast* JL transform of [6], which only requires $\tilde{O}(d + 1/\alpha^2)$ time, a significant speedup in the case where the original dimension d is large.

Corollary 5.6.3. *The same guarantees as in Theorem 5.6.2 holds if we use the fast JL transform and project to $O(\log(1/\alpha)^2/\alpha^2)$ dimensions.*

In the proof of Theorem 5.6.2, we use the following facts about a standard Johnson-Lindenstrauss (JL) projection using Gaussians:

Lemma 5.6.4 ([107, 156])**.** *Let x be a unit vector in $\mathbb{R}^d$ and let G be an (appropriately scaled) Gaussian random projection to k dimensions. Then for $t > 0$:*

$$\Pr(|\|Gx\| - 1| \ge t) \le e^{-t^2 k/8},$$

and

$$\Pr(\|Gx\| \le 1/t) \le \left(\frac{3}{t}\right)^k.$$

Proof of Theorem 5.6.2 . We give the full proof for $f(x, y) = e^{-\|x-y\|_2}$. Carrying out the identical steps with very minor modifications also implies the same statement for $f(x, y) = e^{-\|x-y\|_2^2}$, whose details are omitted. Fix a $x \in X$. We calculate $\mathbb{E}\,|f(x, y) - f(Gx, Gy)|$ (note the randomness is over G). We consider some cases, depending on the value of $f(x, y)$.

Case 1: $f(x, y) \le \alpha$. In this case, if $\|Gx - Gy\|_2 \ge \|x - y\|_2$, then $f(Gx, Gy) \le \alpha$, so the additive error $|f(x, y) - f(Gx, Gy)| \le \alpha$. Thus, the only relevant event is if the distance shrinks, i.e., $\|Gx - Gy\|_2 \le \|x - y\|_2$. If $f(Gx, Gy) \le 3\alpha$ after the projection, then the additive error $|f(x, y) - f(Gx, Gy)| \le O(\alpha)$. Thus, we just have to consider the event $f(Gx, Gy) > 3\alpha$.

For this to happen, we note that $\|x - y\|_2 \ge \log(1/\alpha)$, but $\|Gx - Gy\|_2 \le \log(\alpha^{-1}/3)$. Thus, the distance has shrunk by a factor of

$$\frac{\|x - y\|_2}{\|Gx - Gy\|_2} \ge \frac{\log \alpha^{-1}}{\log(\alpha^{-1}/3)} = \frac{\log(3) + \log(\alpha^{-1}/3)}{\log(\alpha^{-1}/3)} = 1 + \frac{\log(3)}{\log(\alpha^{-1}/3)}.$$

By setting $k = O(\log(1/\alpha)^3)$ and $t = O(1/\log(1/\alpha))$ in Lemma 5.6.4, the probability of this event is at most α, meaning the expected additive error $\mathbb{E}\,|f(x, y) - f(Gx, Gy)|$ can also be bounded by α.

Case 2: $f(x, y) > \alpha$. This is a more involved case, as we need to handle both the sub-cases where the distance increases and decreases. Let $f(x, y) = r > \alpha$.

Sub-case 1: In this sub-case, we bound the probability that $f(Gx, Gy) \le r - \alpha/2$. The original distance is equal to $\log(1/r)$ and the new distance is at least $\log((r - \alpha/2)^{-1})$. The ratio of the new and old distances is $g(r) = \log(r - \alpha/2)/\log(r)$. Writing $r = w\alpha/2$ for $w \ge 2$, we have

$$g(r) = \frac{\log((w-1)\alpha/2)}{\log(w\alpha/2)} = \frac{\log((w-1)/w \cdot w\alpha/2)}{\log(w\alpha/2)} = 1 + \frac{\log(1 - 1/w)}{\log(w\alpha/2)}.$$

As $|\log(1 - 1/w)| = \Theta(1/w)$ for $w \ge 2$, it suffices to upper bound $|w \log(w\alpha/2)|$ in the interval $2 \le w \le 2/\alpha$. One can check that the upper bound occurs for $w = 2/(e\alpha)$, resulting in $\log(1 - 1/w)/\log(w\alpha/2) = \Omega(\alpha)$. Thus by taking $k = O(\log(1/\alpha)/\alpha^2)$ in Lemma 5.6.4, the probability that $f(Gx, Gy) \le r - \alpha/2$ is at most α.

Sub-case 2: In this sub-case, we bound the probability that $f(Gx, Gy) \ge r + \alpha/2$. Again the ratio of the old and new distances is at least $\log(r)/\log(r + \alpha/2)$. Writing $r = w\alpha/2$ for $w \ge 2$, we have

$$\frac{\log(r)}{\log(r + \alpha/2)} = \frac{\log(w\alpha/2)}{\log((w+1)\alpha/2)} = 1 + \frac{\log(1 - 1/(w+1))}{\log((w+1)\alpha/2)}.$$

Thus a similar calculation as above implies that the probability of $f(Gx, Gy) \geq r + \alpha/2$ is at most α by setting $k = O(\log(1/\alpha)/\alpha^2)$ in Lemma 5.6.4.

Altogether, we have bounded the probability of $|f(Gx, Gy) - f(x, y)| \geq \alpha/2$ by at most α, meaning $\mathbb{E}\,|f(Gx, Gy) - G(x, y)| \leq \alpha$, as desired.

Then by linearity and the triangle inequality, it follows that

$$\mathbb{E}\,|z - \hat{z}| \leq \frac{1}{|X|} \sum_x \mathbb{E}\,|f(x, y) - f(Gx, Gy)| \leq \frac{1}{|X|} \sum_x O(\alpha) \leq O(\alpha),$$

as desired. □

We now prove Corollary 5.6.3 where we use the fast JL transform of [6]. However, the fast JL transform, denoted as Π, does not exactly satisfy the concentration bounds of Lemma 5.6.4. In fact, only slightly weaker analogous concentration results are known. Nevertheless, they suffice for our purposes. We quickly review the concentration inequalities known for the fast JL transform and sketch how the proof of Theorem 5.8.2 can be adapted.

Theorem 5.6.5 ([146]). *Let $\Pi : \mathbb{R}^d \to \mathbb{R}^m$ be the fast JL map of [6]. Then for every unit vector $x \in \mathbb{R}^d$, we have:*

1. *If $t \leq \frac{\log m}{\sqrt{m}}$, then*
$$\Pr(|\|\Pi x\|_2^2 - 1| \geq t) \leq e^{-\Omega(\frac{t^2 m}{\log m})}.$$
2. *If $\frac{\log m}{\sqrt{m}} \leq t \leq 1$, then*
$$\Pr(|\|\Pi x\|_2^2 - 1| \geq t) \leq e^{-\Omega(t\sqrt{m})}.$$
3. *If $t \geq 1$, then*
$$\Pr(|\|\Pi x\|_2^2 - 1| \geq t) \leq e^{-\Omega(\sqrt{tm})}.$$

Proof of Corollary 5.6.3. We sketch the modification needed and everything else is identical to the proof of Theorem 5.6.2. Going through the proof, we can check that Case 2 is the only bottleneck that potentially requires a higher projection dimension than Theorem 5.6.2. Here, we need to set $t = \alpha$ in Theorem 5.6.5 and the first inequality there is relevant. However, due to the $\log m$ factor in the denominator, we require an additional $\log(1/\alpha)$ factor in the projection dimension to achieve the same probability of failure as in the proof of Theorem 5.6.2. □

Proof of Theorem 5.6.1. We simply apply our dimensionality reduction result of Corollary 5.6.3 in a black-box manner in conjunction with the data structure of Theorem 5.2.3 from [192]: First we project the datapoints to dimension $\tilde{O}(1/\alpha^2)$ and build the data structure on the projected space. We also release the fast JL projection matrix used which is oblivious of the dataset so it leaks no privacy. Finally, to compute a KDE query, we also project the query vector y using the fast JL projection and query the data structure we built in the lower dimensional space. The construction time, query time, and space all follow from the guarantees of the fast JL transform [6] and Theorem 5.2.3 from [192]. □

5.7 Sparse Function Approximation

We provide details on the function approximation theory, which are later used to obtain our final results on smooth kernels, which are stated in Section 5.8. We use the fact that a small number of exponential sums can approximate smooth kernels, enabling us to reduce this case to prior kernels of Section 5.6. First we recall a classic result.

Theorem 5.7.1 ([171]). *Given $\varepsilon, \delta \in (0,1]$, there exist $O(\log(1/(\varepsilon \cdot \delta)))$ positive numbers $w_j, t_j > 0$, all bounded by $O\left(\frac{1}{\varepsilon \log(1/\delta)}\right)$, such that for all $x \in [\varepsilon, 1]$ we have $(1-\delta)x^{-1} \le \sum_j w_j e^{-t_j x} \le (1+\delta)x^{-1}$. Furthermore, $|w_j e^{-t_j}| \le O(1)$ for all j.*

The theorem implies the following useful corollary.

Corollary 5.7.2. *Given $\alpha \in (0,1]$, there exist $O(\log(1/\alpha))$ positive numbers $w_j, t_j > 0$, all bounded by $O\left(\frac{1}{\alpha \log(1/\alpha)}\right)$, such that for all $x \ge 1$ we have $\left|\sum_j w_j e^{-t_j x} - x^{-1}\right| \le \alpha$.*

Proof. Let $f(x)$ be the approximation to x^{-1} in the interval $[\alpha, 1]$ given by Theorem 5.7.1 for $\delta = O(\alpha)$. Now consider $g(x) = \alpha \cdot f(\alpha x)$. For any $x \in [1, 1/\alpha]$, we have

$$|g(x) - x^{-1}| = |\alpha \cdot f(\alpha x) - x^{-1}| \le |\delta/x| \le O(\alpha)$$

where the first equality follows from the fact that $\alpha \cdot x$ is in the interval $[\alpha, 1]$ for $x \in [1, 1/\alpha]$. Thus, $g(x)$ is an additive $O(\alpha)$ approximation to x^{-1} in the interval $[1, 1/\alpha]$. Now since g and x^{-1} are both decreasing functions of x, and $x^{-1} \le \alpha$ for $x \ge 1/\alpha$, it immediately follows that $g(x)$ is an $O(\alpha)$ additive error approximation for x^{-1} for *all* $x \ge 1$ (note the constant in the O notation has increased). The bounds on the coefficients of g in its exponential sum representation follows from the guarantees of Theorem 5.7.1. $\square$

Using Corollary 5.7.2, we can obtain private KDE data structures for the kernels $f(x,y) = \frac{1}{1+\|x-y\|_2}, \frac{1}{1+\|x-y\|_1}, \frac{1}{1+\|x-y\|_2^2}$ via a black-box reduction to the corresponding private KDE data structures for the kernels $e^{-\|x-y\|_2}, e^{-\|x-y\|_2^2}$, and $e^{-\|x-y\|_1}$.

Theorem 5.7.3. *Let $h(x,y) = \|x-y\|_2, \|x-y\|_2^2$, or $\|x-y\|_1$ and $\alpha \in (0,1)$. Suppose there exists an algorithm for constructing an ε-DP KDE data structure for the kernel $e^{-h(x,y)}$ on a given dataset of size n which answers any query with expected additive error α, takes $C(n,\alpha)$ construction time, $Q(n,\alpha)$ query time, and $S(n,\alpha)$ space, assuming $n \ge L(\varepsilon, \alpha)$.*

Then, there exists an ε-DP data structure for answering KDE queries for $f(x,y) = \frac{1}{1+h(x,y)}$ which answers any query with expected additive error α and the same construction, query, and space as the exponential case, but with n replaced by $O(n \log(1/\alpha))$ and α replaced by $\alpha/\log(1/\alpha)$ in the functions C, Q, S, and L.

Proof. We give a reduction showing how (a small collection of) private KDE data structures for the kernel $e^{-h(x,y)}$ can be used to answer KDE queries for $f(x,y) = \frac{1}{1+h(x,y)}$. Let $g(z)$ be the function guaranteed by Corollary 5.7.2 which approximates $1/z$ by an additive factor for all $z \geq 1$:

$$|\underbrace{\sum_j w_j e^{-t_j z}}_{g(z)} - z^{-1}| \leq O(\alpha) \quad \forall z \geq 1.$$

We have

$$\frac{1}{|X|}\sum_{x \in X} f(x,y) = \frac{1}{|X|}\sum_{x \in X} \frac{1}{1+h(x,y)} = \left(\frac{1}{|X|}\sum_{x \in X}\sum_j w_j e^{-t_j(1+h(x,y))}\right) + O(\alpha)$$

$$= \left[\sum_j w_j e^{-t_j}\left(\frac{1}{|X|}\sum_{x \in X} e^{-t_j h(x,y)}\right)\right] + O(\alpha) = \left[\sum_j w_j e^{-t_j}\left(\frac{1}{|X_j|}\sum_{x \in X_j} e^{-h(x,y_j)}\right)\right] + O(\alpha)$$

where X_j is the dataset $X_j = \{t_j \cdot x \mid x \in X\}$ and y_j is the query $t_j \cdot y$ in the cases that $h(x,y) = \|x-y\|_1$ or $\|x-y\|_2$. In the case where $h(x,y) = \|x-y\|_2^2$ we have $X_j = \{\sqrt{t_j} \cdot x \mid x \in X\}$ and y_j is the query $\sqrt{t_j} \cdot y$.

Note that the function g is public so the parameters w_j and t_j are publicly known (and do not depend on the dataset). Now we simply instantiate private KDE data structures which approximate each of the sums $\frac{1}{|X_j|}\sum_{x \in X_j} e^{-h(x,y)}$. More specifically, we release $O(\log(1/\alpha))$ kernel KDE data structures, one for each X_j, and each of which is $O(\varepsilon/\log(1/\alpha))$-DP. Then the overall data structures we release are ε-DP by composition. Furthermore, since each $w_j e^{-t_j} = O(1)$ and there are only $O(\log(1/\alpha))$ of these terms, if each data structure has expected additive error $O(\alpha/\log(1/\alpha))$, then the overall error is $O(\alpha)$ as well. To summarize, the logarithmic blowup happens in the query/space, as well as any lower-bound assumption on the size of the dataset. □

5.8 New Bounds for Smooth Kernels

In this section, we give new bounds for privately computing KDE queries for the kernels $f(x,y) = \frac{1}{1+\|x-y\|_2}, \frac{1}{1+\|x-y\|_2^2}$, and $\frac{1}{1+\|x-y\|_1}$. Our main result is the following.

Theorem 5.8.1. *Let $\alpha \in (0,1)$ and suppose $n \geq \tilde{O}\left(\frac{1}{\alpha\varepsilon^2}\right)$. For the kernels $f(x,y) = \frac{1}{1+\|x-y\|_2}$ and $f(x,y) = \frac{1}{1+\|x-y\|_2^2}$, there exists an algorithm which outputs an ε-DP data structure with the following properties:*

1. *The expected additive error is α,*
2. *The query time is $\tilde{O}\left(d + \frac{1}{\alpha^4}\right)$,*
3. *The construction time is $\tilde{O}\left(nd + \frac{n}{\alpha^4}\right)$,*
4. *and the space usage is $\tilde{O}\left(d + \frac{1}{\alpha^4}\right)$.*

For the kernel $f(x,y) = \frac{1}{1+\|x-y\|_1}$, we can obtain the following:

1. *The expected additive error is α,*
2. *The query time is $\tilde{O}\left(\frac{d}{\alpha^2}\right)$,*
3. *The construction time is $\tilde{O}\left(\frac{nd}{\alpha^2}\right)$,*
4. *and the space usage is $\tilde{O}\left(\frac{d}{\alpha^2}\right)$.*

The road-map for this section is described in two steps. First, we give new dimensionality reduction results for the first two kernels which obtain the stronger relative error guarantee. Then we show how to combine our dimensionality reduction result with classical function approximation theory to reduce the smooth kernel case to our prior Gaussian and exponential kernel result of Theorem 5.6.1. These results assume a similar condition on n as in our Theorem 5.6.1 and prior works [192]: $n \geq \tilde{O}\left(\frac{1}{\alpha\varepsilon^2}\right)$. We present our novel dimensionality reduction for the kernels $f(x,y) = \frac{1}{1+\|x-y\|_2}$ and $\frac{1}{1+\|x-y\|_2^2}$.

5.8.1 Dimensionality Reduction

Our main result is the following. As before, we assume the projection is chosen independently of the dataset and query.

Theorem 5.8.2 (Dim. Reduction for Smooth Kernels)**.** *Let $G : \mathbb{R}^d \to \mathbb{R}^{1/\alpha^2}$ be a Gaussian JL projection where $\alpha < 1$ is a sufficiently small constant. Fix a query $y \in \mathbb{R}^d$. Let*

$$z = \frac{1}{|X|}\sum_{x\in X} f(x,y),$$
$$\hat{z} = \frac{1}{|X|}\sum_{x\in X} f(Gx,Gy).$$

for $f(x,y) = \frac{1}{1+\|x-y\|_2}$ or $f(x,y) = \frac{1}{1+\|x-y\|_2^2}$. Then, $\mathbb{E}\,|z-\hat{z}| \leq O(\alpha z)$.

A similar corollary as Corollary 5.8.3 also applies to the exponential and Gaussian KDE case.

Corollary 5.8.3. *The same dimensionality reduction bound, up to constant factors, holds as in Theorem 5.8.2, if we use the fast JL transform.*

Proof of Theorem 5.8.2. We give the full proof for $f(x,y) = \frac{1}{1+\|x-y\|_2}$. Carrying out the identical steps with small modifications also implies the same statement for $f(x,y) = \frac{1}{1+\|x-y\|_2^2}$, whose details are omitted. Fix a $x \in X$. We calculate $\mathbb{E}\,|f(x,y) - f(Gx,Gy)|$ (note the randomness is over G). First we consider the case where the distance $\|Gx - Gy\|_2$ expands. Let $\mathcal{A}_i$ be the event that

$$\frac{\|Gx-Gy\|_2 - \|x-y\|_2}{\|x-y\|_2} \in [\alpha i, \alpha(i+1))$$

for $i \geq 0$. We have

$$\begin{aligned}
\sum_{i\geq 0} \Pr[\mathcal{A}_i]\, \mathbb{E}[|f(x,y) - f(Gx,Gy)| \mid \mathcal{A}_i] &\leq \sum_{i\geq 0} e^{-i^2/8} \left(\frac{1}{1+\|x-y\|_2} - \frac{1}{1+\|x-y\|_2(1+\alpha(i+1))} \right) \\
&= \sum_{i\geq 0} e^{-i^2/8} \frac{\|x-y\|_2}{1+\|x-y\|_2} \cdot \frac{\alpha(i+1)}{1+\|x-y\|_2(1+\alpha(i+1))} \\
&\leq \sum_{i\geq 0} e^{-i^2/8} \frac{\alpha(i+1)}{1+\|x-y\|_2} \\
&= \frac{\alpha}{1+\|x-y\|_2} \sum_{i\geq 0} (i+1) e^{-i^2/8} \\
&< \frac{7\alpha}{1+\|x-y\|_2}.
\end{aligned}$$

We now handle the cases where the distance shrinks. We further subdivide this case into sub-cases where the distance shrinks by a factor t satisfying $1 \leq t \leq 6$ and the sub-case where $t \geq 6$. To handle the first sub-case, let $\mathcal{B}_i$ be the event that

$$\frac{\|x-y\|_2}{\|Gx-Gy\|_2} \in [1+\alpha i, 1+\alpha(i+1))$$

for $0 \leq i \leq 5/\alpha$. Note that

$$\begin{aligned}
\mathbb{E}[|f(x,y) - f(Gx,Gy)| \mid \mathcal{B}_i] &\leq \frac{1}{1+\frac{\|x-y\|_2}{(1+\alpha(i+1))}} - \frac{1}{1+\|x-y\|_2} \\
&= \frac{\|x-y\|_2}{\|x-y\|_2+1} \cdot \frac{\alpha(i+1)}{1+\|x-y\|_2+\alpha(i+1)} \\
&\leq \frac{\alpha(i+1)}{1+\|x-y\|_2}.
\end{aligned}$$

Furthermore, under the event $\mathcal{B}_i$, we have that

$$\|x-y\|_2 - \|Gx-Gy\|_2 \geq \left(1 - \frac{1}{1+\alpha i}\right) \|x-y\|_2 \geq \frac{\alpha i}{6} \|x-y\|_2.$$

Thus,

$$\begin{aligned}
\sum_{0\leq i\leq 5/\alpha} \Pr[\mathcal{B}_i]\, \mathbb{E}[|f(x,y) - f(Gx,Gy)| \mid \mathcal{B}_i] &\leq \sum_{0\leq i\leq 5/\alpha} e^{-i^2/288} \cdot \frac{\alpha(i+1)}{1+\|x-y\|_2} \\
&\leq \frac{\alpha}{1+\|x-y\|_2} \sum_{i\geq 0} (i+1) e^{-i^2/288} \\
&< \frac{160\alpha}{1+\|x-y\|_2}.
\end{aligned}$$

For the other sub-case, write it as the union $\cup_{i=1}^{\infty} D_i$ where D_i is the event that

$$3 \cdot 2^{i+1} \geq \frac{\|x-y\|_2}{\|Gx - Gy\|_2} \geq 3 \cdot 2^i,$$

i.e., $\|Gx - Gy\|_2$ shrinks by a factor between $3 \cdot 2^i$ and $3 \cdot 2^{i+1}$. Lemma 5.6.4 gives us

$$\begin{aligned}
\sum_{i \geq 1} \Pr[\mathcal{D}_i] \cdot \mathbb{E}[|f(x,y) - f(Gx,Gy)| \mid \mathcal{D}_i] &\leq \sum_{i \geq 1} \left(\frac{1}{2^i}\right)^k \cdot \left(\frac{1}{1 + \|x-y\|_2/2^{i+1}} - \frac{1}{1+\|x-y\|_2}\right) \\
&\leq \sum_{i \geq 1} \left(\frac{1}{2^i}\right)^{1/\alpha^2} \cdot \frac{2^{i+1}}{1+\|x-y\|_2} \\
&\leq \frac{2}{1+\|x-y\|_2} \cdot \sum_{i \geq 1} \left(\frac{1}{2^i}\right)^{1/\alpha^2 - 1} \\
&\leq \frac{2\alpha}{1+\|x-y\|_2}.
\end{aligned}$$

Together, the above cases imply that

$$\mathbb{E}\,|f(x,y) - f(Gx,Gy)| \leq O(\alpha) \cdot \frac{1}{1+\|x-y\|_2}.$$

Then by linearity and the triangle inequality, it follows that

$$\mathbb{E}\,|z - \hat{z}| \leq \frac{1}{|X|} \sum_x \mathbb{E}\,|f(x,y) - f(Gx,Gy)| \leq O(\alpha) \cdot \frac{1}{|X|} \sum_x \frac{1}{1+\|x-y\|_2} \leq O(\alpha z),$$

as desired. □

Proof of 5.8.3. We outline the steps in the proof of Theorem 5.8.2 which required concentration bounds for the standard JL transform stated in Lemma 5.6.4, and show how they can be appropriately replaced by the guarantees of Theorem 5.6.5. The first step is the sum bounding the contribution of the events $\mathcal{A}_i$, defined as the event where

$$\frac{\|Gx - Gy\|_2 - \|x-y\|_2}{\|x-y\|_2} \in [\alpha i, \alpha(i+1))$$

for $i \geq 0$. Here, for some smaller values of i, the second condition of Theorem 5.6.5 is appropriate and for the rest, the third event is appropriate. Since the sum $\sum_{i \geq 0} i e^{-\sqrt{i}}$ converges to a constant, this portion of the proof carries through. The same comment applies where we bound the contribution of the events $\mathcal{B}_i$. The last place we need to check is when we bound the contribution of the events $\mathcal{D}_i$. Here, the third statement of Theorem 5.6.5 is relevant, and the calculation boils down to showing the sum $\sum_{i \geq 1} e^{-\Omega(\sqrt{3 \cdot 2^i - 1})} \cdot 2^i$ converges to a constant, which is clearly true. □

We are now able to prove Theorem 5.8.1.

Proof of Theorem 5.8.1. The claimed guarantees follow from a simple combination of the tools we have developed so far, and a black-box appeal to the result of Theorem 5.2.3. For the kernel $f(x,y) = \frac{1}{1+\|x-y\|_2}$, we can first perform dimensionality reduction to $O(1/\alpha^2)$ dimensions via an oblivious fast JL projection as stated in Corollary 5.8.3. We then use the reduction given by Theorem 5.7.3 to instantiate $O(\log(1/\alpha))$ copies of private exponential KDE data structure of Theorem 5.2.3. The same procedure works for the kernel $f(x,y) = \frac{1}{1+\|x-y\|_2^2}$. We don't have a dimensionality reduction result for the kernel $f(x,y) = \frac{1}{1+\|x-y\|_1}$, so we just repeat the same steps as above, except we do not perform any dimensionality reduction. The guarantees follow from the guarantees of Theorem 5.2.3 along with the black box reduction given in Theorem 5.7.3. □

5.9 Faster Kernel Density Estimation in the Non-Private Setting

Our novel dimensionality reduction results also obtain faster query algorithms for KDE queries in the *non-private* settings as well in the high-dimensional regime where $d \gg 1/\alpha^2$ where d is the original dimension and α is our desired additive error. Indeed, we can combine our dimensionality reduction bounds of Theorems 5.6.2 and 5.8.2 with any KDE data structure by first projecting to a reduced dimension and then instantiating the KDE data structure in the projected space. Since the dimensionality reduction preserves kernel sums, we can guarantee accurate answers in the projected dimension. In particular, by combining our dimensionality reduction results (the *fast* JL versions of corollaries 5.6.3 and 5.8.3) with prior KDE data structures whose preprocessing and query times are listed in Table 2.1, the following new results easily follow for KDE queries. They give improved query times for the Gaussian, exponential, and the Cauchy kernels. For the Gaussian and exponential kernels, we project to dimension $\tilde{O}(1/\alpha^2)$ where α is the additive error that prior data structures incur and for the Cauchy kernel, we project to dimension $\tilde{O}(1/\varepsilon^2)$, where $1+\varepsilon$ is the multiplicative factors that prior data structures incur.

Theorem 5.9.1. *By combining with [58], for the Gaussian and exponential kernels, we obtain a data structure which gives a $(1+\varepsilon)$ multiplicative and α additive error guarantee for any fixed query with 90% probability with the following preprocessing and query time:*

- *Gaussian kernel: the preprocessing time is $\tilde{O}\left(\frac{nd}{\varepsilon^2\alpha^{0.173+o(1)}}\right)$ and query time $\tilde{O}\left(d + \frac{1}{\varepsilon^2\alpha^{2.173+o(1)}}\right)$.*
- *Exponential kernel: the preprocessing time is $\tilde{O}\left(\frac{nd}{\varepsilon^2\alpha^{0.1+o(1)}}\right)$ and query time $\tilde{O}\left(d + \frac{1}{\varepsilon^2\alpha^{2.1+o(1)}}\right)$.*

For the kernel $\frac{1}{1+\|x-y\|_2^2}$, by combining with [22], we obtain a data structure which gives a $(1+\varepsilon)$ multiplicative error with 90% probability for any fixed query with preprocessing time $\tilde{O}(nd/\varepsilon^2)$ and query time $\tilde{O}(d + \frac{1}{\varepsilon^4})$.

5.10 Empirical Evaluation

We evaluate our algorithms on synthetic and real datasets. We consider three experimental settings which together are representative of our main upper-bound results. We show the average of 20 trials and ± 1 standard deviation is shaded where appropriate. All experiments unless stated otherwise are implemented in Python 3.9 on an M1 MacbookPro with 32GB of RAM.

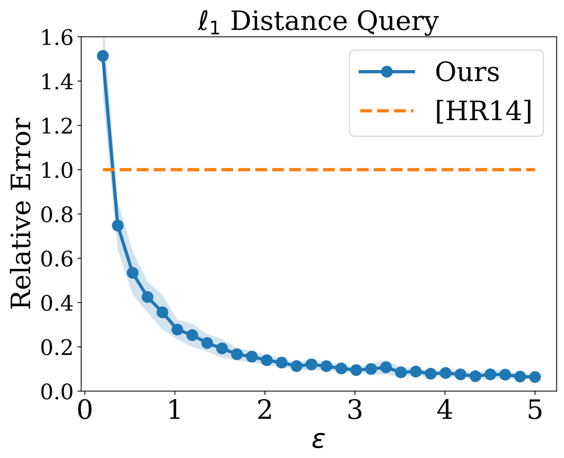
5.10.1 ℓ_1 Experiments.

The task here is to approximate the (normalized) map $y \to \frac{1}{n}\sum_{x \in X} |x-y|$ for a one dimensional dataset X of size $n = 10^3$, with points randomly picked in $[0,1]$. The query points are $O(n)$ evenly spaced points in $[0,1]$. We implement our one-dimensional ℓ_1 distance query algorithm and compare to the prior baseline of [102]. Both our and [102]'s high-dimensional algorithms are constructed by instantiating d different copies of one-dimensional data structures (on the standard coordinates). Thus, the performance in one dimension is directly correlated with the high-dimensional case. In our case, the map we wish to approximate converges to $\int_0^1 |x-y|\, dx = y^2 - y + 1/2$ for $y \in [0,1]$, allowing us to easily compare to the ground truth. In Figure 5.1a, we plot the average relative error across all queries as a function of ε. The explicit parameter settings for the algorithm given in [102] are extremely large in practice, meaning the output of their algorithm was always the identically 0 function, which gives relative error equal to 1 (the distance query was always estimated to be 0) for the values of ε tested, as shown in Figure 5.1a. On the other hand, our algorithm gives non-trivial empirical performance and its error decreases smoothly as ε increases. Indeed, Figure 5.1b shows our output values (scaled by $1/n$) for various ε's. We can observe that our estimates converge to the true function as ε increases. We observed qualitatively similar results for both algorithms for the larger $n = 10^6$ case as well.

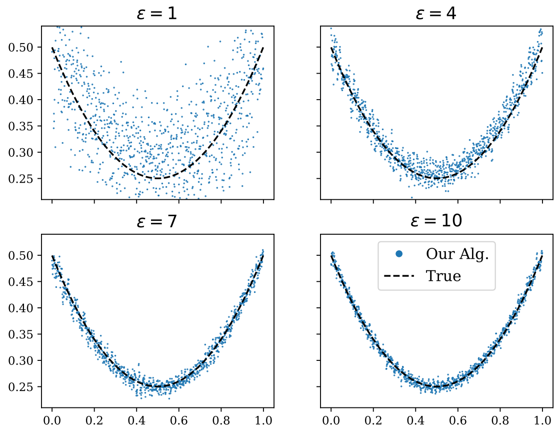
5.10.2 Dimensionality Reduction Experiments.

We empirically demonstrate that dimensionality reduction provides computational savings for DP-KDE without significantly degrading accuracy. Our task is to approximate KDE values for the Gaussian kernel $e^{-\|x-y\|_2^2}$. We compare against the prior SOTA [192]. Our provable dimensionality reduction result of Theorem 5.6.2 gives a general framework: apply an oblivious dimensionality reduction to the data and use any DP-KDE algorithm in the projected space. Indeed, Theorem 5.6.1 follows by applying the framework to the prior SOTA algorithm of [192]. Thus in our experiment, we use the randomized dimension reduction of Theorem 5.6.2 in conjunction with the implementation of [192]. Note that while we fix the DP-KDE implementation used after dimensionality reduction, our framework is compatible with any other choice and we expect qualitatively similar results with other choices.

Our dataset consists of embeddings of CIFAR-10 in dimension 2048, computed from an Inception-v3 model [186], pre-trained on ImageNet [73]. Obtaining embeddings of private



(a) Our relative error decreases as ε increases.



(b) Our performance for various fixed ε.

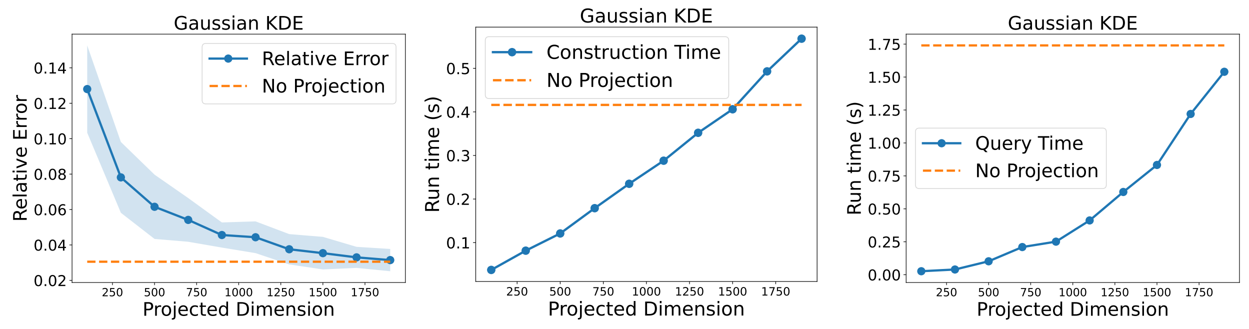
Figure 5.1: Our algorithm for ℓ_1 queries has smaller error than prior SOTA of [102].

datasets from pre-trained ML models is standard in the applied DP literature [72, 203]. The intuition is that the embeddings from the network are powerful enough to faithfully represent the images in Euclidean space, so computing kernel values on these features is meaningful. We project the embeddings to lower dimensions d ranging from 200 to 2000. We use the training points of a fixed label as the private dataset and the corresponding test set as the queries.

Figure 5.2a shows the relative error of our approach (in blue) and the baseline of [192] (in orange) which does not use any dimensionality reduction. The relative errors of both are computed by comparing to the ground truth. Figure 5.2b shows the construction time of the private data structure and Figure 5.2c shows the total query time on the test points. We see that the relative error smoothly decreases as we project to more dimensions, while construction and query time smoothly increase. Note the construction time includes the time to compute the projection. For example, projecting to $d = 1000$ increases the relative error by 0.015 in absolute terms, while reducing the construction time by ≈ 2x and reducing the construction time by a factor of > 4x.

5.10.3 DP Classification.

We consider the DP classification task on Cifar-10. The train and test splits are the private data and query respectively, and the task is to train a private classifier on the train set to classify the test set. Our methodology is extremely simple, fast, and does not require any specialized hardware like GPUs: we instantiate an (ε, δ)-DP distance query data structure on each class. The classes disjointly partition the data so the output, which is an (ε, δ)-DP data structure for each class, is overall (ε, δ)-DP [163]. We use $f(x, y) = \|x - y\|_2^2$ since it has the simplest algorithm (see Section 5.5.1). It essentially reduces to releasing a noisy mean for every class. To classify a query, we assign it to the class whose (noisy) mean is closest to the



(a) Proj. dimension vs rel. error (b) Construction time (c) Total query time

Figure 5.2: Results for our dimensionality reduction experiments.

query. For other potential choices of f like kernels, one would instead assign to the class with the highest KDE value. There are two competing SOTA works: one from Deepmind [72] and [203]. We first give a high-level methodology of prior works: they start with a pre-trained model on ImageNet[3] and fine tune using DP-gradient descent/SGD. Note that the vectors we build our private data structures on are the penultimate layer embeddings of the ResNet pre-trained model used in [203]. Thus, all methods have the access to the same 'pre-training' information.

Note a simple but important point: ε, δ are input parameters, so we cannot just output the data structure or ML model with the highest accuracy. The data structure or model we output must satisfy the given privacy guarantee. Thus, accuracy vs privacy vs runtime are non-trivial trade-offs. The full details are given below.

Methodology of [203]. We use the "GP" baseline from [203], which trains a linear classifier with DP-SGD [4] on top of features from SimCLR [61]. Deviating from the vanilla DP-SGD, GP uses all samples to compute gradients at every iteration (i.e., no subsampling) as it was found to perform better. In our implementation, we use the "r152_2x_sk1" SimCLR network released from [61] to extract the features of the images. When training the linear classifier, we do a grid search of the hyper-parameters (learning rate $\in [0.1, 0.05, 0.1]$, gradient clipping threshold $\in [1.0, 0.1, 0.01]$, noise multiplier $\in [100, 500, 1000]$) and take the best combination. Following the common practice [203], we ignore the privacy cost of this hyper-parameter tuning process.

Methodology of [72]. [72] pre-train the WRN-28-10 network [207] on ImageNet and fine-tunes it on CIFAR10 with DP-SGD [4]. We use their official code for the experiments. We do a grid search of the noise multiplier ($\in [9.4, 12.0, 15.8, 21.1, 25.0]$) where the first four values are used in the paper and the last value is an additional one we test. We report the

[3]The works consider other alternate datasets, but we only compare to the Imagenet case. We expect qualitatively similar results when pre-training with other datasets.

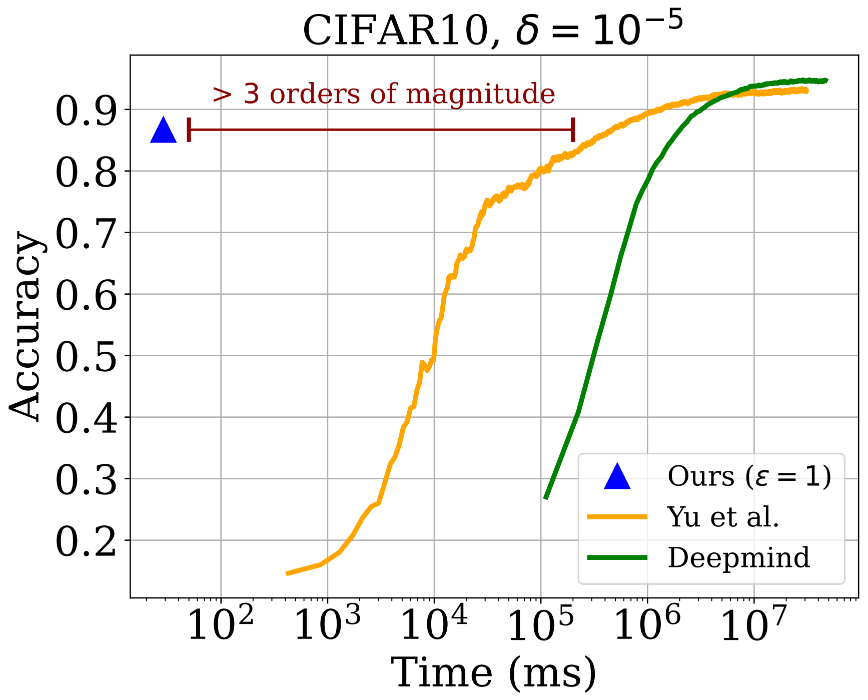


Figure 5.3: Runtime vs Accuracy.

best results across these values and ignore the privacy cost of this hyper-parameter tuning process.

Our hyper-parameters. For our method, we take the embeddings from the pre-trained "r152_3x_sk1" SimCLR network released from [61]. Our embeddings were in dimensions 6144. Since we are computing the ℓ_2 distance squared, we can apply Johnson-Lindenstrauss to reduce the dimensionality, without any privacy loss. Furthermore, we can clip the embeddings as well, which reduces the overall sensitivity of our algorithm (to reduce the R dependency in Section 5.5.1) Thus, our hyper-parameters were the projection dimensions, which we looped from 100 to 2000 and the clipping threshold, which we picked from 10 choices in $[0.001, 1]$.

Let us temporarily ignore δ for simplicity of discussion. If they use T steps of training, every model update step approximately satisfies $\varepsilon/\sqrt{T}$ privacy (exact bounds depend on the DP composition formulas), ensuring the overall final model is (ε, δ)-DP. Thus, every step for them incurs some privacy budget, with the benefit of increased accuracy. Therefore, these works can stop training at intermediate times to obtain a model with stronger privacy guarantees (lower ε), but worse accuracy. However, the same procedure also gives an accuracy vs model training time trade-off for these prior works. This is shown in Figure 5.3. The right most endpoints of both baselines (the largest times), correspond to models with $\varepsilon = 1$. In other words, their models with the worst privacy guarantees has the highest accuracy, while also requiring the longest training time.

In contrast, our algorithm of Section 5.5.1 simply releases a noisy collection of fixed vectors. Our data structure construction *time*, which corresponds to their model training time, is independent of (ε, δ) (but *accuracy* depends on them). In Figure 5.3, we plot the accuracy of our data structure for $\varepsilon = 1$ (we use the best hyper-parameter choices for all methods). For other values of ε, we would simply incur the same construction time, but observe differing accuracy since the construction time is independent of ε (but again accuracy improves for higher ε). The time to initialize our data structure (for all classes) is 28.8 ms *on a CPU*, and the query time for all queries was 73.8 ms. On the other hand, fully training the model of [203] up to $\varepsilon = 1$ takes $>$ 8.5 hours on a single NVIDIA RTX A6000 GPU. The runtimes of [72] are even longer since they use larger models. Thus, as shown in Figure 5.3, creating our private data structure is $>$ **3 orders of magnitude** faster than the time to create models of corresponding accuracy via the two baselines. Note that we are also using arguably inferior hardware. The best accuracy of all methods as a function of ε, ignoring run-time considerations, is shown in Figure 5.4.

Additional Results. In Figure 5.4, we also show the ε vs accuracy trade-off, ignoring runtime. We plot accuracy as a function of the privacy ε. δ is always 10^{-5}. We also plot the best performance of every tested method: we iterate over the hyper-parameters of all methods including both [72] and [203] using their code, and we display the best accuracy for every value of ε. Hyper-parameters are described above. Note the trivial accuracy is .1 via random labels. We see that for small values of ε, we obtain the second best results, but lag behind both prior SOTA for large ε regime. The accuracy in the large $\varepsilon \geq 1$ regime are $0.87, 0.93, 0.95$ respectively for ours, [203], and [72]. However, our approach has a major run-time benefit compared to these prior works, as argued in Section 5.10. Such a boost in runtime may justify the drop in accuracy in some applications.

Note that the main bottleneck in accuracy of our method is the quality of embeddings used. If we ignore all privacy constraints, then our average similarity based methodology obtains accuracy close to 0.87 without accounting for privacy. This is very close to the performance obtained by our $\varepsilon = 1$ private data structure of Figure 5.4. Thus, we cannot hope to do better in terms of accuracy. However, our method is extremely flexible in the sense that better initial embeddings, for example from other models or models pre-trained on additional or different data, can automatically lead to better downstream performance.

5.11 Conclusion

We give improved theoretical algorithms for computing similarity to private datasets for a wide range of functions f. Our algorithms have the added benefit of being practical to implement. We view our results as the tip of the iceberg in understanding similarity computations on private datasets. Many exciting open directions remain such as obtaining improved upper bounds or showing lower bounds for the f's we considered. It is also an interesting direction to derive algorithms for more complicated 'similarity' measures, such as Optimal Transport (OT), although it is not clear what notion of privacy we should

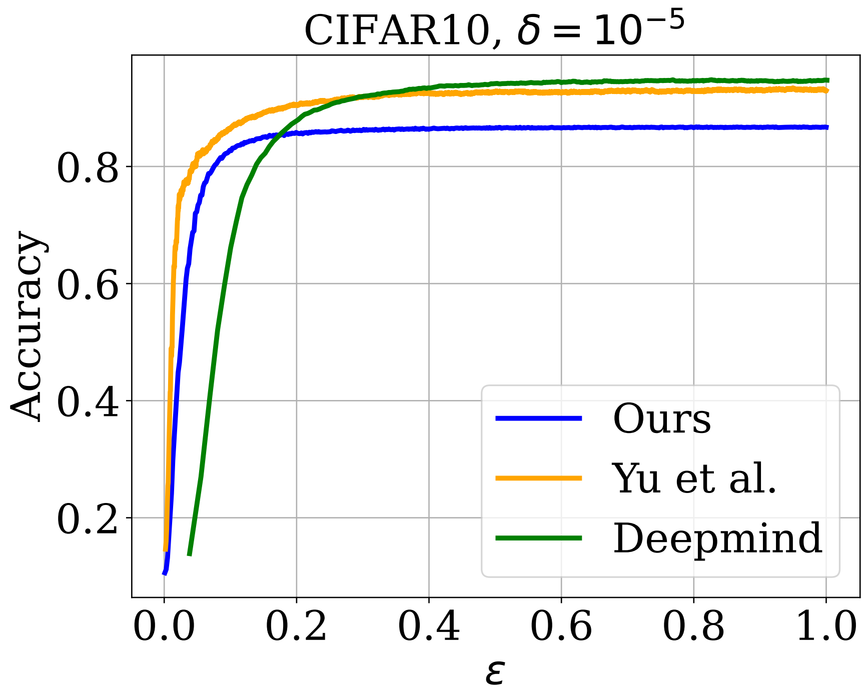


Figure 5.4: Accuracy of all methods as a function of ε, ignoring all run-time constraints. The best hyper-parameter choices are used for all methods.

use for such measures. Lastly, generalizing our proof-of-concept experiment on DP image classification to text or other domains, using embeddings computed from models such as LLMs, is also an interesting empirical direction.

Part II

Efficient 'Global' Similarity Analysis

Chapter 6

Subquadratic Algorithms for Distance Matrices

In this chapter, we give subquadratic algorithms for analyzing distance matrices. Recall that given a set of n points $X = \{x_1, \ldots, x_n\}$, the distance matrix of X with respect to a distance function f is defined as the $n \times n$ matrix A satisfying $A_{i,j} = f(x_i, x_j)$. The main results of this chapter are centered around the following three broad categories.

> *1. We study upper and lower bounds for constructing matrix-vector queries for a wide array of distance matrices.*

A matrix-vector query algorithm accepts a vector z as input and outputs the vector Az. There is substantial motivation for studying such queries. Indeed, there is now a rich literature for fundamental linear algebra algorithms which are in the "matrix free" or "implicit" model. These algorithms only assume access to the underlying matrix via matrix-vector queries. Some well known algorithms in this model include the power method for computing eigenvalues and the conjugate gradient descent method for solving a system of linear equations. For many fundamental functions of A, nearly optimal bounds in terms of the number of queries have been achieved [29, 40, 154]. Furthermore, having access to matrix-vector queries also allows the simulation of any randomized sketching algorithm, a well studied algorithmic paradigm in its own right [201]. This is because randomized sketching algorithms operate on the matrix ΠA or $A\Pi$ where Π is a suitably chosen random matrix, such as a Gaussian matrix. Typically, Π is chosen so that the sketches ΠA or $A\Pi$ have significantly smaller row or column dimension compared to A. If A is symmetric, we can easily acquire both types of matrix sketches via a small number of matrix-vector queries.

Therefore, creating efficient versions of matrix-vector queries for distance matrices automatically lends itself to many further downstream applications. We remark that our algorithms can access to the set of input points but *do not* explicitly create the distance matrix. A canonical example of our upper bound results is the construction of matrix-vector queries for the function $f(x, y) = \|x - y\|_p^p$.

Theorem 1.5.1. *Let $p \geq 1$ be an integer. Suppose we are given a dataset of n points $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$. X implicitly defines the matrix $A_{i,j} = \|x_i - x_j\|_p^p$. Given a query $z \in \mathbb{R}^n$, we can compute Az exactly in time $O(ndp)$. If p is odd, we also require $O(nd \log n)$ preprocessing time.*

We give similar guarantees for a wide array of functions f and we refer the reader to Table 6.1 which summarizes our matrix-vector query upper bound results. Note that some of the functions f we study in Table 6.1 do not necessarily induce a metric in the strict mathematical sense (for example the function $f(x, y) = \|x - y\|_2^2$ does not satisfy the triangle inequality). Nevertheless, we still refer to such functions under the broad umbrella term of “distance functions" for ease of notation. We always explicitly state the function f we are referring to.

Crucially, most of our bounds have a linear dependency on n which allows for scalable computation as the size of the dataset X grows. Our upper bounds are optimal in many cases, see Theorem 6.2.12.

Function	$f(x,y)$	Preprocessing	Query Time	Reference
ℓ_p^p for p even	$\|x-y\|_p^p$	–	$O(ndp)$	Thm 6.2.1 / 6.2.3
ℓ_p^p for p odd	$\|x-y\|_p^p$	$O(nd\log n)$	$O(ndp)$	Thm 6.1.2 / 6.2.4
Mixed ℓ_∞	$\max_{i,j} \lvert x_i - y_j \rvert$	$O(nd\log n)$	$O(n^2)$	Thm 6.2.5
Mahalanobis Distance^2	$x^T M y$	$O(nd^2)$	$O(nd)$	Thm 6.2.6
Polynomial Kernel	$\langle x, y\rangle^p$	–	$O(nd^p)$	Thm 6.2.7
Total Variation Distance	$\mathrm{TV}(x,y)$	$O(nd\log n)$	$O(nd)$	Thm 6.2.8
KL Divergence	$\mathrm{D}_{\mathrm{KL}}(x \,\|\, y)$	–	$O(nd)$	Thm 6.2.2
Symmetric Divergence	$\mathrm{D}_{\mathrm{KL}}(x \,\|\, y) + \mathrm{D}_{\mathrm{KL}}(y \,\|\, x)$	–	$O(nd)$	Thm 6.2.9
Cross Entropy	$H(x,y)$	–	$O(nd)$	Thm 6.2.9
Hellinger Distance^2	$\sum_{i=1}^d \sqrt{x(i)y(i)}$	–	$O(nd)$	Thm 6.2.10

Table 6.1: A summary of our results for exact matrix-vector queries.

Combining our upper bound results with optimized matrix-free methods, immediate corollaries of our results include faster algorithms for eigenvalue and singular value computations and low-rank approximations. Low-rank approximation is of special interest as it has been widely studied for distance matrices; for low-rank approximation, our bounds outperform prior results for specific distance functions. For example, for the ℓ_1 and ℓ_2^2 case (and in general PSD matrices), [28] showed that a rank-k approximation can be found in time $O(ndk/\varepsilon + nk^{w-1}/\varepsilon^{w-1})$. This bound has extra $\mathrm{poly}(1/\varepsilon)$ overhead compared to our bound stated in Table 6.2. The work of [113] has a worse $\mathrm{poly}(k, 1/\varepsilon)$ overhead for an additive error approximation for the ℓ_2 case. See Section 6.0.1 for further discussion of prior works. The downstream applications of matrix-vector queries are summarized in Table 6.2.

We also study fundamental limits for any upper bound algorithms. In particular, we show that *no algorithm* can compute a matrix-vector query for general inputs for the ℓ_∞ metric in subquadratic time, assuming a standard complexity-theory assumption called the *Strong Exponential Time Hypothesis (SETH)* [103, 104].

Theorem 1.5.2. *For any $\alpha > 0$ and $d = \omega(\log n)$, any algorithm for exactly computing Az for any input z, where A is the ℓ_∞ distance matrix, requires $\Omega(n^{2-\alpha})$ time (assuming SETH).*

This shows a separation between the functions listed in Table 6.1 and the ℓ_∞ metric. Surprisingly, we can create queries for the *approximate* matrix-vector query in substantially faster time.

Theorem 6.0.1. *Suppose $X \subseteq \{0, 1, \dots, O(1)\}^d$. We can compute By in time $O(n \cdot d^{O(\sqrt{d}\log(d/\varepsilon))})$ where $\|A - B\|_\infty \le \varepsilon$.*

To put the above result into context, the lower bound of Theorem 1.5.2 holds for points sets in $\{0, 1, 2\}^d$ in $d \approx \log n$ dimensions. In contrast, if we relax to an approximation guarantee, we can obtain a subquadratic-time algorithm for d up to $\Theta(\log^2(n)/\log\log(n))$.

Finally, we provide a general understanding of the limits of our upper bound techniques. In Theorem 6.3.1, we show that essentially the only f for which our upper bound techniques apply have a "linear structure" after a suitable transformation. We refer to Appendix Section 6.3 for details.

Problem	$f(x, y)$	Runtime	Prior Work
$(1+\varepsilon)$ Relative error rank k low-rank approximation	ℓ_1, ℓ_2^2	$\tilde{O}\left(\frac{ndk}{\varepsilon^{1/3}} + \frac{nk^{w-1}}{\varepsilon^{(w-1)/3}}\right)$ Theorem 6.5.4	$O\left(\frac{ndk}{\varepsilon} + \frac{nk^{w-1}}{\varepsilon^{w-1}}\right)$ [28]
Additive error $\varepsilon\|A\|_F$ rank k low-rank approximation	ℓ_2	$\tilde{O}\left(\frac{ndk}{\varepsilon^{1/3}} + \frac{nk^{w-1}}{\varepsilon^{(w-1)/3}}\right)$ Theorem 6.5.6	$\tilde{O}(nd \cdot \text{poly}(k, 1/\varepsilon))$ [113]
$(1+\varepsilon)$ Relative error rank k low-rank approximation	Any in Table 6.1	$\tilde{O}\left(\frac{Tk}{\varepsilon^{1/3}} + \frac{nk^{w-1}}{\varepsilon^{(w-1)/3}}\right)$ Theorem 6.5.7	$\tilde{O}\left(\frac{n^2dk}{\varepsilon^{1/3}} + \frac{nk^{w-1}}{\varepsilon^{(w-1)/3}}\right)$ [29]
$(1 \pm \varepsilon)$ Approximation to top k singular values	Any in Table 6.1	$\tilde{O}\left(\frac{Tk}{\varepsilon^{1/2}} + \frac{nk^2}{\varepsilon} + \frac{k^3}{\varepsilon^{3/2}}\right)$ Theorem 6.5.8	$\tilde{O}\left(\frac{n^2dk}{\varepsilon^{1/2}} + \frac{nk^2}{\varepsilon} + \frac{k^3}{\varepsilon}^{3/2}\right)$ [154]
Multiply distance matrix A with any $B \in \mathbb{R}^{n \times n}$	Any in Table 6.1	$O(Tn)$ Lemma 6.5.9	$O(n^w)$
Multiply two distance matrices A and B	ℓ_2^2	$O(n^2d^{w-2})$ Lemma 6.5.11	$O(n^w)$

Table 6.2: Applications of our matrix-vector query results. T denotes the matrix-vector query time, given in Table 6.1. $w \approx 2.37$ is the matrix multiplication constant.

2. We give algorithms for multiplying distance matrices faster than general matrices.

Fast matrix-vector queries also automatically imply fast matrix multiplication, which can be reduced to a series of matrix-vector queries. For concreteness, if f is the ℓ_p^p function which induces A, and B is any $n \times n$ matrix, we can compute AB in time $O(n^2dp)$. This is substantially faster than the general matrix multiplication bound of $n^w \approx n^{2.37}$. We also give an improvement of this result for the case where we are multiplying two distance matrices arising from ℓ_2^2. See Table 6.2 for summary.

3. We give fast algorithms for constructing distance matrices.

Finally, we give fast algorithms for constructing approximate distance matrices. To establish some context, recall the classical Johnson-Lindenstrauss (JL) lemma which (roughly) states that a random projection of a dataset $X \subset \mathbb{R}^d$ of size n onto a dimension of size $O(\log n)$ approximately preserves all pairwise distances [119]. A common applications of this lemma is to *instantiate* the ℓ_2 distance matrix. A naive algorithm which computes the distance matrix after performing the JL projection requires approximately $O(n^2 \log n)$ time. Surprisingly, we show that the JL lemma is not tight with respect to creating an approximate ℓ_2 distance matrix; we show that one can initialize the ℓ_2 distance in an asymptotically better runtime.

Theorem 1.5.3. (Informal; See Theorem 6.6.5) *We can calculate a $n \times n$ matrix B such that each (i, j) entry B_{ij} of B satisfies $(1 - \varepsilon)\|x_i - x_j\|_2 \le B_{ij} \le (1 + \varepsilon)\|x_i - x_j\|_2$ in time $O(\varepsilon^{-2}n^2 \log^2(\varepsilon^{-1} \log n))$.*

Our result can be viewed as the natural runtime bound which would follow if the JL lemma implied an embedding dimension bound of $O(\text{poly}(\log \log n))$. While this is impossible, as it would imply an exponential improvement over the JL bound which is tight [131], we achieve our speedup by carefully reusing distance calculations via tools from metric compression [111]. Our results also extend to the ℓ_1 distance matrix; see Theorem 6.6.5 for details.

Notation. For points in X, we denote $x_i(j)$ to be the jth coordinate of point x_i for clarity. For all other vectors v, v_i denotes the ith coordinate. We are interested in matrices of the form $A_{i,j} = f(x_i, x_j)$ for $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ which measures the similarity between any pair of points. f might not necessarily be a distance function but we use the terminology "distance function" for ease of notation. We will always explicitly state the function f as needed. $w \approx 2.37$ denotes the matrix multiplication constant, i.e., the exponent of n in the time required to compute the product of two $n \times n$ matrix [8].

6.0.1 Related Works

Matrix-Vector Products Queries. Our work can be understood as being part of a long line of classical works on the matrix free or implicit model as well as the active recent line of works on the matrix-vector query model. Many widely used linear algebraic algorithms such as the power method, the Lanczos algorithm [129], conjugate gradient descent [175], and Wiedemann's coordinate recurrence algorithm [196], to name a few, all fall into this paradigm. Recent works such as [29, 40, 154] have succeeded in precisely nailing down the query complexity of these classical algorithms in addition to various other algorithmic tasks such as low-rank approximation [29], trace estimation [150], and other linear-algebraic functions [169, 185]. There is also a rich literature on query based algorithms in other contexts with the goal of minimizing the number of queries used. Examples include graph queries [88], distribution queries [45], and constraint based queries [79] in property testing, inner product queries in compressed sensing [78], and quantum queries [66, 135].

Most prior works on query based models assume black-box access to matrix-vector queries. While this is a natural model which allows for the design non-trivial algorithms and lower bounds, it is not always clear how such queries can be initialized. In contrast, the focus of our work is not on obtaining query complexity bounds, but rather complementing prior works by creating an efficient matrix-vector query for a natural class of matrices.

Subquadratic Algorithms for Distance Matrices. Most work on subquadratic algorithms for distance matrices have focused on the problem of computing a low-rank approximation. [26, 113] both obtain an additive error low-rank approximation applicable for all distance matrices. These works only assume access to the entries of the distance matrix whereas we assume we also have access to the underlying dataset. [28] study the problem of computing the low-rank approximation of PSD matrices with also sample access to the entries of the matrix. Their results extend to low-rank approximation for the ℓ_1 and ℓ_2^2 distance matrices in addition to other more specialized metrics such as spherical metrics. Table 6.2 lists the runtime comparisons between their results and ours.

Practically, the algorithm of [113] is the easiest to implement and has outstanding empirical performance. We note that we can easily simulate their algorithm with no overall asymptotic runtime overhead using $O(\log n)$ vector queries. Indeed, their algorithm proceeds by sampling rows of the matrix according to their ℓ_2^2 value and then post-processing these rows. The sampling probabilities only need to be accurate up to a factor of two. We can acquire these sampling probabilities by performing $O(\log n)$ matrix-vector queries which sketches the rows onto dimension $O(\log n)$ and preserves all row-norms up to a factor of two with high probability due to the Johnson-Lindenstrauss lemma [119]. This procedure only incurs an additional runtime of $O(T \log n)$ where T is the time required to perform a matrix-vector query.

The paper [110] shows that the exact L_1 distance matrix can be created in time $O(n^{(w+3)/2}) \approx n^{2.69}$ in the case of $d = n$, which is asymptotically faster than the naive bound of $O(n^2 d) = O(n^3)$. In contrast, we focus on creating an (entry-wise) approximate distance matrices for all values of d.

We also compare to the paper of [9]. In summary, their main upper bounds are approximation algorithms while we mainly focus on exact algorithms. Concretely, they study matrix vector products for matrices of the form $A_{i,j} = f(\|x_i - x_j\|_2^2)$ for some function $f : \mathbb{R} \to \mathbb{R}$.

They present results on approximating the matrix vector product of A where the approximation error is additive. They also consider a wide range of f, including polynomials and other kernels, but the input to is always the ℓ_2 distance squared. In contrast, we also present exact algorithms, i.e., with no approximation errors. For example one of our main upper bounds is an exact algorithm when $A_{i,j} = \|x_i - x_j\|_1$ (see Table 1 for the full list). Since it is possible to approximately embed the ℓ_1 distance into ℓ_2^2, their methods could be used to derive approximate algorithms for ℓ_1, but not the exact ones. Furthermore, we also study a wide variety of other distance functions such as ℓ_∞ and ℓ_p^p (and others listed in Table 1) which are not studied in Alman et al. In terms of technique, the main upper bound technique of Alman et al. is to expand $f(\|x_i - x_j\|_2^2)$ and approximate the resulting quantity via a polynomial. This is related to our upper bound results for ℓ_p^p for even p where we also use polynomials. However, our results are exact, while theirs are approximate. Our ℓ_1 upper bound technique is orthogonal to the polynomial approximation techniques used in Alman et al. We also employ polynomial techniques to give upper bounds for the approximate ℓ_∞ distance function which is not studied in Alman et al. Lastly, Alman et al. also focus on the Laplacian matrix of the weighted graph represented by the distance matrix, such as spectral sparsification and Laplacian system solving. In contrast, we study different problems including low-rank approximations, eigenvalue estimation, and the task of initializing an approximate distance matrix. We do not consider the distance matrix as a graph or consider the associated Laplacian matrix.

It is also easy to verify the "folklore" fact that for a gram matrix AA^T, we can compute AA^Tv in time $O(nd)$ if $A \in \mathbb{R}^{n\times d}$ by computing A^Tv first and then $A(A^Tv)$. Our upper bound for the ℓ_2^2 function can be reduced to this folklore fact by noting that $\|x-y\|_2^2 = \|x\|_2^2 + \|y\|_2^2 - 2\langle x, y\rangle$. Thus the ℓ_2^2 matrix can be decomposed into two rank one components due to the terms $\|x\|_2^2$ and $\|y\|_2^2$, and a gram matrix due to the term $\langle x, y\rangle$. This decomposition of the ℓ_2^2 matrix is well-known (see Section 2 in [74]). Hence, a matrix-vector query for the ℓ_2^2 matrix easily reduces to the gram matrix case. Nevertheless, we explicitly state the ℓ_2^2 upper bound for completeness since we also consider all ℓ_p^p functions for any integer $p \geq 1$.

Polynomial Kernels. There have also been works on faster algorithms for approximating a kernel matrix K defined as the $n \times n$ matrix with entries $K_{i,j} = k(x_i, x_j)$ for a kernel function k. Specifically for the polynomial kernel $k(x_i, x_j) = \langle x_i, x_j\rangle^p$, recent works such as [5, 19, 180, 199] have shown how to find a sketch K' of K which approximately satisfies $\|K'z\|_2 \approx \|Kz\|_2$ for all z. In contrast, we can exactly simulate the matrix-vector product Kz. Our runtime is $O(nd^p)$ which has a linear dependence on n but an exponential dependence on p while the aforementioned works have at least a quadratic dependence on n but a polynomial dependence on p. Thus our results are mostly applicable to the setting where our dataset is large, i.e. $n \gg d$ and p is a small constant. For example, $p = 2$ is a common choice in practice [54]. Algorithms with polynomial dependence in d and p but quadratic dependence in n are suited for smaller datasets which have very large d and large p. Note that a large p might arise if approximates a non-polynomial kernel using a polynomial kernel via a taylor expansion. We refer to the references within [5, 19, 180, 199] for additional related work. There is also work on kernel density estimation (KDE) data structures which upon query y, allow for estimation of the sum $\sum_{x \in X} k(x, y)$ in time sublinear in $|X|$ after some preprocessing on the dataset X. For widely used kernels such as the Gaussian and Laplacian kernels, KDE data structures were used in [24] to create a matrix-vector query algorithm for kernel matrices in time subquadratic in $|X|$ for input vectors which are entry wise non-negative. We refer the reader to [22, 23, 55, 58, 177] and references within for prior works on KDE data structures.

6.1 Faster Matrix-Vector Product Queries for ℓ_1

We derive faster matrix-vector queries for distance matrices for a wide array of distance metrics. First we consider the case of the ℓ_1 metric such that $A_{i,j} = f(x_i, x_j)$ where $f(x, y) = \|x - y\|_1 = \sum_{i=1}^{d} |x_i - y_i|$.

Algorithm 8 Preprocessing

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: procedure PREPROCESSING
3:     for $i \in [d]$ do
4:         $T_i \leftarrow$ sorted array of the $i$th coordinates of all $x \in X$.
5:     end for
6: end procedure
```

Algorithm 9 matrix-vector Query for $p = 1$

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: Output: $z = Ay$
3: procedure QUERY($\{T_i\}_{i\in[d]}$, $y$)
4:     $y_1, \cdots, y_n \leftarrow$ coordinates of $y$.
5:     Associate every $x_i \in X$ with the scalar $y_i$
6:     for $i \in [d]$ do
7:         Compute two arrays $B_i, C_i$ as follows.
8:         $B_i$ contains the partial sums of $y_j x_j(i)$ computed in the order induced by $T_i$
9:         $C_i$ contains the partial sums of $y_j$ computed in the order induced by $T_i$
10:    end for
11:    $z \leftarrow 0^n$
12:    for $k \in [n]$ do
13:        for $i \in [d]$ do
14:            $q \leftarrow$ position of $x_k(i)$ in the order of $T_i$
15:            $S_1 \leftarrow B_i[q]$
16:            $S_2 \leftarrow B_i[n] - B_i[q]$
17:            $S_3 \leftarrow C_i[q]$
18:            $S_4 \leftarrow C_i[n] - C_i[q]$
19:            $z(k) += x_k(i) \cdot (S_3 - S_4) + S_2 - S_1$
20:        end for
21:    end for
22: end procedure
```

We first analyze the correctness of Algorithm 9.

Theorem 6.1.1. *Let $A_{i,j} = \|x_i - x_j\|_1$. Algorithm 9 computes Ay exactly.*

Proof. Consider any coordinate $k \in [n]$. We show that $(Ay)_k$ is computed exactly. We have

$$(Ay)(k) = \sum_{j=1}^{n} y_j \|x_k - x_j\|_1 = \sum_{j=1}^{n}\sum_{i=1}^{d} y_j |x_k(i) - x_j(i)| = \sum_{i=1}^{d}\sum_{j=1}^{n} y_j |x_k(i) - x_j(i)|.$$

Let π^i denote the order of $[n]$ induced by T_i. We have

$$\sum_{i=1}^{d}\sum_{j=1}^{n} y_j |x_k(i) - x_j(i)| = \sum_{i=1}^{d}\left(\sum_{j:\pi^i(k)\le\pi^i(j)} y_j(x_j(i) - x_k(i)) + \sum_{j:\pi^i(k)>\pi^i(j)} y_j(x_k(i) - x_j(i))\right).$$

We now consider the inner sum. It rearranges to the following:

$$x_k(i)\left(\sum_{j:\pi^i(k)>\pi^i(j)} y_j - \sum_{j:\pi^i(k)\le\pi^i(j)} y_j\right) + \sum_{j:\pi^i(k)\le\pi^i(j)} y_j x_j(i) - \sum_{j:\pi^i(k)>\pi^i(j)} y_j x_j(i)$$
$$= x_k(i)\cdot(S_3 - S_4) + S_2 - S_1,$$

where S_1, S_2, S_3, and S_4 are defined in lines $15-18$ of Algorithm 9 and the last equality follows from the definition of the arrays B_i and C_i. Summing over all $i \in [d]$ gives us the desired result. $\square$

The following theorem readily follows.

Theorem 6.1.2. *Suppose we are given a dataset $\{x_1, \ldots, x_n\}$ which implicitly defines the distance matrix $A_{i,j} = \|x_i - x_j\|_1$. Given a query $y \in \mathbb{R}^d$, we can compute Ay exactly in $O(nd)$ query time. We also require a one time $O(nd \log n)$ preprocessing time which can be reused for all queries.*

6.2 General Upper Bounds for Faster Matrix-Vector Queries

We now consider the case of ℓ_p^p for $p = 2$. Generalizing the results of $p = 1$ and $p = 2$ allows us to handle general ℓ_p^p functions.

Algorithm 10 matrix-vector Query for $p = 2$

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: Output: $z = Ay$
3: procedure QUERY(y)
4:     $v \leftarrow \sum_{i=j}^n y_j x_j$
5:     $S_1 \leftarrow \sum_{i=j}^n y_j^2$
6:     $S_2 \leftarrow \sum_{i=j}^n y_j^2 \|x\|_2^2$
7:     $z \leftarrow 0^n$
8:     for $k \in [n]$ do
9:         $z(k) \leftarrow S_1 \|x_k\|_2^2 + S_2 - 2\langle x_k, v \rangle$
10:    end for
11: end procedure
```

Theorem 6.2.1. *We can compute Ay in $O(nd)$ query time.*

Proof. The proof follows from the following calculation of the kth coordinate of Ay:

$$(Ay)(k) = \sum_{j=1}^n y_j \|x_k - x_j\|_2^2 = \|x_k\|_2^2 \left(\sum_{j=1}^n y_j^2 \right) + \sum_{j=1}^n y_j^2 \|x_j\|_2^2 - 2 \left\langle x_k, \sum_{j=1}^n y_j x_j \right\rangle . \qquad \square$$

We can extend our results to general ℓ_p^p functions as well as a wide array of commonly used functions to measure (dis)similarity between vectors. For example, suppose the points x_i represent a probability distribution over the domain $[n] := \{1, \ldots, n\}$. A widely used "distance" function over distributions is the KL-divergence defined as

$$f(x_i, x_j) = \mathrm{D_{KL}}(x_i \,\|\, x_j) = \sum_{k \in [d]} x_i(k) \log(x_i(k)) - x_i(k) \log(x_j(k)) = -H(x_i) - \sum_{k \in [d]} x_i(k) \log(x_j(k)),$$

where H is the entropy function. Our techniques extend to the KL-divergence as well.

Algorithm 11 matrix-vector Query for KL Divergence

```
1: **Input:** Dataset $X \subset \mathbb{R}^d$
2: **Output:** $z = Ay$
3: **procedure** QUERY( $y$)
4:     $S_i \leftarrow \sum_{j=1}^{n} y_j \log(x_j(i))$ for all $i \in [d]$
5:     $H_i \leftarrow H(x_i)$ for all $i \in [n]$
6:     $Y \leftarrow \sum_{j=1}^{n} y_j$
7:     $z \leftarrow 0^n$
8:     **for** $k \in [n]$ **do**
9:         $z(k) \leftarrow -H_k \cdot Y - \sum_{i=1}^{d} x_k(i) S_i$
10:    **end for**
11: **end procedure**
```

Theorem 6.2.2. *We can compute Ay in $O(nd)$ query time.*

Proof. Note that computed all of S_i and H_i takes $O(nd)$ time. Now

$$\begin{aligned}
(Ay)(k) &= \sum_{j=1}^{n} y_j \mathrm{D}_{\mathrm{KL}}(x_k \,\|\, x_j) \\
&= \sum_{j=1}^{n} -y_j H(x_k) - \sum_{j=1}^{n} y_j \sum_{k=1}^{d} x_i(k) \log(x_j(k)) \\
&= -H(x_k) \left(\sum_{j=1}^{n} y_j \right) - \sum_{k=1}^{d} \sum_{j=1}^{n} y_j x_i(k) \log(x_j(k)) \\
&= -H_k \cdot Y - \sum_{k=1}^{d} x_i(k) S_k,
\end{aligned}$$

as desired. □

6.2.1 General p

We now consider the case of a general non-negative even integer p.

Algorithm 12 matrix-vector Query for even p

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: Output: $z = Ay$
3: procedure QUERY(y)
4:     Compute all the values $S_{i,t} := \sum_{j=1}^{n} x_j(i)^{p-t}(-1)^{p-t}$ for all $i \in [d]$ and $t \in \{0, \dots, p\}$
5:     $z \leftarrow 0^n$
6:     for $k \in [n]$ do
7:         $z(k) \leftarrow \sum_{i=1}^{d} \sum_{t=1}^{p} x_k(i)^t S_{i,t}$
8:     end for
9: end procedure
```

Theorem 6.2.3. *We can compute Ay in $O(ndp)$ query time.*

Proof. Consider the following calculation of the kth coordinate of Ay:

$$\begin{aligned}(Ay)(k) &= \sum_{j=1}^{n} y_j \|x_k - x_j\|_p^p \\ &= \sum_{j=1}^{n} y_j \sum_{i=1}^{d} (x_k(i) - x_j(i))^p \\ &= \sum_{j=1}^{n} \sum_{i=1}^{d} \sum_{t=1}^{p} x_k(i)^t x_j(i)^{p-t}(-1)^{p-t} \\ &= \sum_{i=1}^{d} \sum_{t=1}^{p} x_k(i)^t \sum_{j=1}^{n} x_j(i)^{p-t}(-1)^{p-t} \\ &= \sum_{i=1}^{d} \sum_{t=1}^{p} x_k(i)^t S_{i,t}.\end{aligned}$$

Note that computing $S_{i,t}$ for all i and t takes $O(ndp)$ time. Then returning the value of $(Ay)_k$ takes $O(dp)$ time resulting in the claimed runtime. □

The case of a general non-negative odd integer p follows in a straightforward manner by combining the above techniques with those of the $p = 1$ case of Theorem 6.1.2 so we omit the proof.

Theorem 6.2.4. *For odd integer p, we can compute Ay in $O(nd \log n)$ preprocessing time and $O(ndp)$ query time.*

6.2.2 Other Distance Functions

In this section we initialize matrix-vector queries for a wide variety of “distance" functions.

'Mixed' ℓ_∞. We consider the case of a 'permutation invariant' version of the ℓ_∞ norm defined as follows:

$$f(x, y) = \max_{i \in [d], j \in [d]} |x_i - y_j|.$$

f is not a norm but we will refer to it as 'mixed' ℓ_∞.

Algorithm 13 Preprocessing

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: procedure PREPROCESSING
3:     for $j \in [n]$ do
4:         $\min_j, \max_j \leftarrow$ minimum and maximum values of the entries of $x_j$, respectively.
5:     end for
6: end procedure
```

Algorithm 14 matrix-vector Query for mixed ℓ_∞

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: Output: $z = Ay$
3: procedure QUERY($\{\min_j, \max_j\}_{j=1}^n, y$)
4:     $z \leftarrow 0^n$
5:     for $k \in [n]$ do
6:         $z(k) \leftarrow \sum_{j=1}^n y_j \cdot \max(|\min_k - \min_j|, |\min_k - \max_j|, |\max_k - \min_j|, |\max_k - \max_j|)$
7:     end for
8: end procedure
```

Theorem 6.2.5. *We can compute Ay in $O(nd)$ preprocessing time and $O(n^2)$ query time.*

Proof. The preprocessing time holds because we calculate the maximum and minimum of a list of d numbers a total of n times. For the query time, note that each $z(k)$ takes $O(n)$ time to compute since we do a $O(1)$ operation is each index of the sum in Line 6 of Algorithm 14.

To prove correctness, note that for any two vectors $x, y \in \mathbb{R}^d$, the maximum value of $|x_i - y_j|$ is attained if x_i and y_j are among the minimum / maximum values of the coordinates of x and y respectively. To see this, fix a value of x_i. We can always increase $|x_i - y_j|$ by setting y_j to be the maximum or minimum over all j. $\square$

Mahalanobis Distance Squared. We consider the function

$$f(x, y) = x^T M y$$

for some $d \times d$ matrix M. This is the squared version of the well-known Mahalanobis distance.

Algorithm 15 Preprocessing

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: procedure PREPROCESSING
3:     $S \leftarrow d \times n$ matrix where the $j$th column is $Mx_j$ for all $j \in [n]$.
4: end procedure
```

Algorithm 16 matrix-vector Query for Mahalanobis distance squared

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: Output: $z = Ay$
3: procedure QUERY( $S, y$)
4:     $v \leftarrow Sy$
5:     $z \leftarrow 0^n$
6:     for $k \in [n]$ do
7:         $z(k) \leftarrow \langle x_k, v\rangle$
8:     end for
9: end procedure
```

Theorem 6.2.6. *We can compute Ay in $O(nd^2)$ preprocessing time and $O(nd)$ query time.*

Proof. Note that the kth coordinate of Ay is given by

$$(Ay)(k) = \sum_{j=1}^{n} y_j x_k^T M x_j = \left\langle x_k, \sum_{j=1}^{n} y_j M x_k \right\rangle = \langle x_k, Sy\rangle$$

which proves correctness. It takes $O(nd^2)$ time to compute S, $O(nd)$ time to compute Sy, and then $O(d)$ time to compute the kth coordinate of Ay for all $k \in [n]$. $\square$

Polynomial Kernels. We now consider polynomial kernels of the form

$$f(x, y) = \langle x, y\rangle^p.$$

Theorem 6.2.7. *We can compute Ay in $O(nd^p)$ query time.*

Proof Sketch. Consider the following expression

$$g(z) = \sum_{j=1}^{n} y_j \langle z, x_j\rangle^p$$

as a polynomial $g : \mathbb{R}^d \to \mathbb{R}$ in the d coordinates of z. By rearranging, the above sum can be written as a sum over $O(d^p)$ terms, corresponding to each monomial $z_1^{a_1} \dots z_d^{a_d}$ where $a_1 + \dots + a_d = p$. The coefficient of each term takes $O(nd)$ time to compute given x_i and y. Once computed, we can evaluate the polynomial at $z = x_j$ for all j which form the coordinates of Ay. Again, this can be viewed as “linearizing" the kernel given by $\langle x, y\rangle^p$. $\square$

We note that a proof similar to that of Theorem 6.2.7 was given in Section 5.3 of [9] by expanding the relevant quantity as a polynomial; see Section 6.0.1 for detailed comparison between [9] and our work.

6.2.3 Distances for Distributions

We now consider the case that each x_i specifies a discrete distribution over a domain of d elements. Matrices A where $A_{i,j}$ is a function computing distances between distributions x_i and x_j have recently been studied in machine learning.

We consider how to construct matrix-vector queries for such matrices for a range of widely used distance measures on distributions. First note that a result on the TV distance follows immediately from our ℓ_1 result.

Theorem 6.2.8. *Suppose $A_{i,j} = \mathrm{TV}(x_i, x_j)$. We can compute Ay in $O(nd \log n)$ preprocessing time and $O(nd)$ query time.*

We now consider some other distance functions on distributions.

Divergences. Through a similar calculation as the KL divergence case, we can also achieve $O(nd)$ query times if f is the Jensen-Shannon divergence, defined as

$$f(x,y) = \frac{\mathrm{D}_{\mathrm{KL}}(x \,\|\, y) + \mathrm{D}_{\mathrm{KL}}(y \,\|\, x)}{2},$$

as well as the cross entropy function.

Theorem 6.2.9. *Let f be the Jensen-Shannon divergence or cross entropy function. Then Ay can be computed in $O(nd)$ time.*

Through a similar calculation as done in Section 6.2.2 (for the case of $p = 1$), we can also perform matrix-vector multiplication queries in the case that

$$f(x,y) = \sum_{i=1}^{d} \sqrt{x(i)y(i)}.$$

This is the squared Hellinger distance.

Theorem 6.2.10. *Let f be the squared Hellinger distance. Then Ay can be computed in $O(nd)$ time.*

6.2.4 Approximate Matrix-Vector Query for ℓ_2

While our techniques do not extend to the ℓ_2 case for *exact* matrix-vector queries, we can nonetheless instantiate *approximate* matrix-vector queries for the ℓ_2 function. We recall the following well known embedding result given in Theorem 2.0.2

We can instantiate approximate matrix-vector queries for $f(x, y) = \|x - y\|_2$ via the following algorithm.

Algorithm 17 Preprocessing

```
1: Input: Dataset $X \subset \mathbb{R}^d$
2: procedure PREPROCESSING($T$)
3:     $X' \leftarrow TX$ where $T$ is the linear map from Theorem 2.0.2
4:     Run Algorithm 8 on $X'$
5: end procedure
```

For queries, we just run Algorithm 9 on X'. We have the following guarantee:

Theorem 6.2.11. *Let $A_{i,j} = \|x_i - x_j\|_2$. There exists a matrix B such that we can compute By in $O(nd^2 + nd \log n)$ preprocessing time and $O(n \log(n)/\varepsilon^2)$ query time where*

$$\|A - B\|_F \le \varepsilon \|A\|_F$$

with probability $1 - 1/\text{poly}(n)$.

Proof. The preprocessing and query time follow from the time required to apply the transformation T from Theorem 2.0.2 to our set of points X as well as the time needed for the ℓ_1 matrix-vector query result of Theorem 6.1.1. The Frobenius norm guarantee follows from the fact that every entry of A will be approximated with relative error in B using Theorem 2.0.2. □

6.2.5 Matrix-Vector Query Lower Bounds

Table 6.1 shows that we can initialize matrix-vector queries for a variety of distance functions in $O(nd)$ time. It is straightforward to see that this bound is optimal for a large class of distance matrices.

Theorem 6.2.12. *Consider the case that $A_{i,j} = f(x_i, x_j)$ satisfying $f(x, x) = 0$ for all x. Further assume that for all x, there exists an input y such that $f(x, y) = 1$. An algorithm which outputs an entry-wise approximation of Az to any constant factor for input z requires $\Omega(nd)$ time in the worst case.*

Proof. We consider two cases for input points of A. In the first case, all points in our dataset X are identical. In the second case, a randomly chosen point is distance 1 away from the $n - 1$ identical points. Computing the product of A times the all ones vector allows us to distinguish the two cases as $A1$ has entries summing to 0 in the first case whereas $A1$ has entries summing to $n - 1$ in the second case. Thus to approximate $A1$ entry-wise to any constant factor, we must distinguish the two cases. If we read $o(n)$ points, then with good probability we will see no duplicates. Thus, we must read $\Omega(n)$ points, require $\Omega(nd)$ time. □

6.3 When Do Our (Exact) Upper Bound Techniques Work?

By this point, we have seen many examples of matrix-vector queries which can be initialized as well as a lower bound for a natural distance function which prohibits any subquadratic time algorithm. Naturally, we are thus interested in the limits of our upper bound techniques for instantiating fast matrix-vector product. An understanding of such limits sheds light on families of structured matrices which may admit fast matrix-vector queries in general. In this section we fully characterize the capabilities of our upper bound methods and show that essentially our techniques can only work "linear" functions (in a possibly different basis).

First we set some notation. Let A be a $n \times n$ matrix we wish to compute where each (i, j) entry is given by $f(x_i, x_j)$. Given a query vector $z \in \mathbb{R}^n$, the kth coordinate of Az is given by

$$(Az)(k) = \sum_{i=1}^{n} z_i f(x_k, x_i).$$

An example choice of f is given by $f(x, y) = \sum_{j=1}^{d} x(j) \log(y(j))$ (assuming all the coordinates of x and y are entry wise non-negative. Note this is related to the cross entropy function in Table 6.1).

We first highlight the major steps which are common to all of our upper bound algorithms using f as an example. Our upper bound technique proceeds as follows:

- Break up $f(x, y)$ into a sum over d terms: $\sum_{j=1}^{d} x(j) \log(y(j))$.
- Switch the order of summation:
$$(Az)(k) = \sum_{i=1}^{n} z_i f(x_k, x_i) = \sum_{j=1}^{d} \sum_{i=1}^{n} z_i x_k(j) \log(x_i(j)).$$
- Evaluate each of the inner d summations with 1 evaluation each (after some preprocessing). In other words, for a fixed j, each of the sums $\sum_{i=1}^{n} z_i x_k(j) \log(x_i(j))$ can be computed as one evaluation, namely $x_k(j) \cdot \left(\sum_{i=1}^{n} z_i \log(x_i(j)) \right)$ and in preprocessing, we can compute $\sum_{i=1}^{n} z_i \log(x_i(j))$ as it does not depend on the coordinate k.

The key steps of the above outline, namely switching the order of summation and precomputation of repeated terms, can be encapsulated in the following framework.

Theorem 6.3.1. *Suppose there exist mappings $T_1, T_2 : \mathbb{R}^d \to \mathbb{R}^{d'}$ (possibly non-linear) and* *a continuous $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that for every k,*

$$\begin{aligned}(Az)(k) &= \sum_{i=1}^{n} z_i f(x_k, x_i) \\ &= \sum_{i=1}^{n} z_i \sum_{j=1}^{d'} g\left(T_1(x_k)(j), T_2(x_i)(j)\right) \quad \text{(breaking } f \text{ into sum over } d' \text{ terms)} \\ &= \sum_{j=1}^{d'} \sum_{i=1}^{n} z_i \cdot g\left(T_1(x_k)(j), T_2(x_i)(j)\right) \quad \text{(switching order of summation).}\end{aligned}$$

Further suppose that each of the terms $\sum_{i=1}^{n} z_i \cdot g\left(T_1(x_k)(j), T_2(x_i)(j)\right)$ can be evaluated as

$$\sum_{i=1}^{n} z_i \cdot g\left(T_1(x_k)(j), T_2(x_i)(j)\right) = g\left(T_1(x_k)(j), \sum_{i=1}^{n} z_i T_2(x_i)(j)\right)$$

for any choice of the vector z. Then $g(a, b)$ must be a linear function in b.

Theorem 6.3.1 is stated in quite general terms. We are stipulating the following statements: the functions T_1, T_2 represent possibly non-linear transformations to $\mathbb{R}^{d'}$ on x, y respectively such that $f(x, y)$ can be decomposed as a sum over d' function evaluations. Each function evaluation takes in the same coordinate, say the jth coordinate, of both $T_1(x)$ and $T_2(y)$ and computes the function $g(T_1(x)(j), T_2(y)(j))$. Finally the resulting sum $\sum_{i=1}^{n} z_i \cdot g\left(T_1(x_k)(j), T_2(x_i)(j)\right)$ can be computed as $g\Big(T_1(x_k)(j), \sum_{i=1}^{n} h(z_i) T_2(x_i)(j)\Big)$.

If these conditions hold (which is precisely the case in the proof of *all* our upper bound results), then it *must* be the case that g has a very special form, in particular, g must be a linear function in its second variable. To make the setting more concrete, we map the terminology of Theorem 6.3.1 into some examples from our upper bound results.

First consider the case that $f(x, y) = \langle x, y \rangle$. In this case, both T_1 and T_2 are the identity maps and $g(a, b) = ab$. It is indeed the case that $g(a, b)$ is linear in b. Now consider a slightly more complicated choice $f(x, y) = \sum_{j=1}^{d} x(j) \log(y(j))$. Here, we first have the mappings $T_1 =$ identity but T_2 is a coordinate wise map such that $T_2(y) = [\log(y_1), \dots, \log(y_n)]$. The function g again satisfies $g(a, b) = ab$. Finally we consider the example $f(x, y) = \|x - y\|_2^2$ which sets $d' \gg d$. In particular, the mappings T_1, T_2 expand x, y into a $O(d^2)$-dimensional vector, whose coordinates represent all possible combinations products of two coordinates of x and y respectively. (The reader may realize that this particular case is an example of "linearizing" the kernel given by f). Again g is the same function as before.

The proof of Theorem 6.3.1 relies on the following classical result on the solutions of Cauchy's functional equation.

Theorem 6.3.2. *Let $t : \mathbb{R} \to \mathbb{R}$ be a continuous function which satisfies $t(x+y) = t(x)+t(y)$ for all inputs x, y in its domain. Then t must be a linear function.*

For us the hypothesis that t is continuous suffices but it is know that the above result follows from much weaker hypothesis. We refer to the reader to [126] for reference related to Cauchy's functional equation.

Proof of Theorem 6.3.1. Our goal is to show that if

$$\sum_{i=1}^{n} z_i \cdot g\left(T_1(x_k)(j), T_2(x_i)(j)\right) = g\left(T_1(x_k)(j), \sum_{i=1}^{n} z_i T_2(x_i)(j)\right)$$

for all z and choices of input points x_i then g must be linear in the second variable. First set $z_j = 0$ for all $j \geq 2$ and $z_1 = z_2 = 1$. For ease of notation, denote $q := T_1(x_k)(j)$. As we vary the coordinate of the points x_1 and x_2, the values $T_2(x_1)(j)$ and $T_2(x_2)(j)$ also vary over the range of T_2. Thus,

$$g(q, a) + g(q, b) = g(q, a + b)$$

for all possible inputs a, b. However, this is exactly the hypothesis of Theorem 6.3.2 so it follows that g must be a linear function in its second coordinate, as desired. $\square$

While the proof of Theorem 6.3.1 is straightforward, it precisely captures the scenarios where our upper bound techniques apply. In short, it implies that f must have a linear structure, under a suitable change of basis, for our techniques to hold. If its not the case, then our techniques do not apply and new ideas are needed. Nevertheless, as displayed by the versatility of examples in Table 6.1, such a structure is quite common in many applications where matrices of distance or similarity functions arise.

The observant reader might wonder how our result for the ℓ_1 function fits into the above framework as it is not obviously linear. However, we note that the function $h_j(x) = \sum_{i=1}^n |x(j) - x_i(j)|$ (which appears in the theorem statement of Theorem 6.3.1 as the sum $\sum_{i=1}^n z_i \cdot g\Big(T_1(x_k)(j), T_2(x_i)(j)\Big)$ is actually a *piece-wise* linear function in $x(j)$. The sorting preprocessing we performed for Theorem 6.1.1 can be thought of as creating a data structure which allows us to efficiently index into the correct linear piece.

6.4 Lower and (Approximate) Upper bounds for ℓ_∞

In this section we give a proof of Theorem 1.5.2. Specifically, we give a reduction from the *Orthogonal Vector Problem* (OVP) [197] to the problem of computing matrix-vector product Az, where $A_{i,j} = \|x_i - x_j\|_\infty$, for a given set of points $X = \{x_1, \dots, x_n\}$. Recall the orthogonal vector problem and the associated conjecture.

Definition 2.0.2. (Orthogonal Vectors problem (OVP)) *Given two sets of vectors $A = \{a^1, \dots, a^n\}$ and $B = \{b^1, \dots, b^n\}$, $A, B \subset \{0,1\}^d$, $|A| = |B| = n$, determine whether there exist $x \in A$ and $y \in B$ such that the dot product $x \cdot y = \sum_{j=1}^d x_j y_j$ (taken over reals) is equal to 0.*

Conjecture 2.0.2. [197] *Suppose $d = \omega(\log n)$. Assuming SETH, for every constant $\alpha > 0$, no randomized algorithm can solve OVP in $O(n^{2-\alpha})$ time.*

An efficient reduction from OVP to the matrix-vector product problem yields Theorem 1.5.2.

Lemma 6.4.1. *If the matrix-vector product problem for ℓ_∞ distance matrices induced by n vectors of dimension d can be solved in time $T(n,d)$, then OVP (with the same parameters) can be solved in time $O(T(n,d))$.*

Proof. Define two functions, $f, g : \{0,1\}^d \to [0,1]$, such that $f(0) = g(0) = 1/2$, $f(1) = 0$, $g(1) = 1$. We extend both functions to vectors by applying f and g coordinate wise and to sets by letting $f(\{a^1, \dots, a^n\}) = \{f(a^1), \dots, f(a^n)\}$; the function g is extended in the same way for B. Observe that, for any pair of non-zero vectors $a, b \in \{0,1\}^d$, we have $\|f(a) - g(b)\|_\infty = 1$ if and only if $a \cdot b > 0$, and $\|f(a) - g(b)\|_\infty = 1/2$ otherwise.

Consider two sets of binary vectors A and B. Without loss of generality we can assume that the vectors are non-zero, since otherwise the problem is trivial. Define three distance matrices: matrix M_A defined by the set $f(A)$, matrix M_B defined by the set $g(B)$ and M_{AB} defined by the set $f(A) \cup f(B)$. Furthermore, let M be the "cross-distance" matrix, such that such that $M_{i,j} = \|f(a^i) - g(b^j)\|_\infty$. Observe that the matrix M_{AB} contains blocks M_A and M_B on its diagonal, and blocks M and M^T off-diagonal. Thus, $M_{AB} \cdot 1 = M_A \cdot 1 + M_B \cdot 1 + 2M \cdot 1$, where 1 denotes an all-ones vector of the appropriate dimension. Since $M \cdot 1 = (M_{AB} \cdot 1 - M_A \cdot 1 - M_B \cdot 1)/2$, we can calculate $M \cdot 1$ in time $O(T(n,d))$. Since all entries of M are either 1 or 1/2, we have that $M \cdot 1 < n^2$ if and only if there is an entry $M_{i,j} = 1/2$. However, this only occurs if $a^i \cdot b^j = 0$. □

6.4.1 Approximate ℓ_∞ Matrix-Vector Queries

In light of the lower bounds given above, we consider initializing *approximate* matrix-vector queries for the ℓ_∞ function. Note that the lower bound holds for points in $\{0,1,2\}^d$ and thus it is natural to consider approximate upper bounds for the case of limited alphabet.

Binary Case. We first consider the case that all points $x \in X$ are from $\{0,1\}^d$. We first claim the existence of a polynomial T with the following properties. Indeed, the standard Chebyshev polynomials satisfy the following lemma, see e.g., see Chapter 2 in [172].

Lemma 6.4.2. *There exists a polynomial $T : \mathbb{R} \to \mathbb{R}$ of degree $O(\sqrt{d}\log(1/\varepsilon))$ such that $T(0) = 0$ and $|T(x) - 1| \le \varepsilon$ for all $x \in [1/d, 1]$.*

Now note that $\|x-y\|_\infty$ can only take on two values, 0 or 1. Furthermore, $\|x-y\|_\infty = 0$ if and only if $\|x-y\|_2^2 = 0$ and $\|x-y\|_\infty = 1$ if and only if $\|x-y\|_2^2 \ge 1$. Therefore, $\|x-y\|_\infty = 0$ if and only if $T(\|x-y\|_2^2/d) = 0$ and $\|x-y\|_\infty = 1$ if and only if $|T(\|x-y\|_2^2/d) - 1| \le \varepsilon$. Thus, we have that

$$|A_{i,j} - T(\|x_i - x_j\|_2^2/d)| = |\|x_i - x_j\|_\infty - T(\|x_i - x_j\|_2^2/d)| \le \varepsilon$$

for all entries $A_{i,j}$ of A. Note that $T(\|x-y\|_2^2/d)$ is a polynomial with $O((2d)^t)$ monomials in the variables $x(1), \ldots, x(d)$. Consider the matrix B satisfying $B_{i,j} = T(\|x_i - x_j\|_2^2/d)$. Using the same ideas as our upper bound results for $f(x,y) = \langle x, y\rangle^p$, it is straightforward to calculate the matrix vector product By (see Section 6.2.2). To summarize, for each $k \in [n]$, we write the kth coordinate of By as a polynomial in the d coordinates of x_k. This polynomial has $O((2d)^t)$ monomials and can be constructed in $O(n(2d)^t)$ time. Once constructed, we can evaluate the polynomial at $x_1, \ldots, x_n$ to obtain all the n coordinates of By. Each evaluation requires $O((2d)^t)$ resulting in an overall time bound of $O(n(2d)^t)$.

Theorem 6.4.3. *Let $A_{i,j} = \|x_i - x_j\|_\infty$. We can compute By in time $O(n(2d)^{\sqrt{d}\log(1/\varepsilon)})$ where $\|A - B\|_\infty \le \varepsilon$.*

Entries in $\{0, \ldots, M\}$. We now consider the case that all points $x \in X$ are from $\{0, \ldots, M\}^d$. Our argument will be a generalization of the previous section. At a high level, our goal is to detect which of the $M+1$ possible values in $\{0, \ldots, M\}$ is equal to the ℓ_∞ norm. To do so, we appeal to the prior section and design estimators which approximate the indicator function "$\|x-y\|_\infty \ge i$". By summing up these indicators, we can approximate $\|x-y\|_\infty$.

Our estimators will again be designed using the Chebyshev polynomials. To motivate them, suppose that we want to detect if $\|x-y\|_\infty \ge i$ or if $\|x-y\|_\infty < i$. In the first case, some entry in $x-y$ will have absolute value value at least i where as in the other case, all entries of $x-y$ will be bounded by $i-1$ in absolute value. Thus if we can boost this 'signal', we can apply a polynomial which performs thresholding to distinguish the two cases. This motivates considering the functions of $\|x-y\|_k^k$ for a larger power k. In particular, in the case that $\|x-y\|_\infty \ge i$, we have $\|x-y\|_k^k \ge i^k$ and otherwise, $\|x-y\|_k^k \le di^{k-1}$. By setting $k \approx \log(d)$, the first value is much larger than the latter, which we can detect using the 'threshold' polynomials of the previous section.

We now formalize our intuition. It is known that appropriately scaled Chebyshev polynomials satisfy the following guarantees, see e.g., see Chapter 2 in [172].

Lemma 6.4.4. *There exists a polynomial $T : \mathbb{R} \to \mathbb{R}$ of degree $O(\sqrt{t}\log(t/\varepsilon))$ such that $|T(x)| \le \varepsilon/t$ for all $x \in [0, 1/(10t)]$ and $|T(x) - 1| \le \varepsilon/t^2$ for all $x \in [1/t, 1]$.*

Given $x, y \in \mathbb{R}^d$, our estimator will first try to detect if $\|x-y\|_\infty \ge i$. Let T_1 be a polynomial from Lemma 6.4.4 with $t = O(M^k)$ for $k = O(M\log(Md))$ and assuming k is even. Let T_2 be a polynomial from Lemma 6.4.4 with $t = O(\sqrt{d}\log(M/\varepsilon))$. Our estimator will be

$$T_2\left(\frac{1}{d}\sum_{j=1}^{d} T_1\left(\frac{(x(j)-y(j))^k}{i^k \cdot M^k}\right)\right).$$

If coordinate j is such that $|x(j) - y(j)| \ge i$, then

$$\frac{(x(j)-y(j))^k}{i^k \cdot M^k} \ge \frac{1}{M^k}$$

and so T_1 will evaluate to a value very close to 1. Otherwise, we know that

$$\frac{(x(j)-y(j))^k}{i^k \cdot M^k} \le \frac{(i-1)^k}{i^k M^k} = \frac{1}{M^k}\left(1-1/i\right)^k \ll \frac{1}{M^k} \cdot \frac{1}{\text{poly}(M,d)}$$

by our choice of k, which means that T_1 will evaluate to a value close to 0. Formally,

$$\frac{1}{d}\sum_{j=1}^{d} T_1\left(\frac{(x(j)-y(j))^k}{i^k \cdot M^k}\right)$$

will be at least $1/d$ if there is a $j \in [d]$ with $|x(j)-y(j)| \ge i$ and otherwise, will be at most $1/(10d)$. By our choice of T_2, the overall estimate output at least $1-\varepsilon$ in the first case and a value at most ε in the second case.

The polynomial which is the concatenation of T_2 and T_1 has $O\left((dk \cdot \deg(T_1))^{\deg(T_2)}\right) = (dM)^{O(M\sqrt{d}\log(Md))}$ monomials, if we consider the expression as a polynomial in the variables $x(1), \ldots, x(d)$. Our final estimator will be the sum across all $i \ge 1$. Following our upper bound techniques for matrix-vector products for polynomial, e.g. in Section 6.2.2, and as outlined in the prior section, we get the following overall query time:

Theorem 6.4.5. *Suppose we are given $X = \{x_1, \ldots, x_n\} \subseteq \{0, \ldots, M\}^d$ which implicitly defines the matrix $A_{i,j} = \|x_i - x_j\|_\infty$. For any query y, we can compute By in time $n \cdot (dM)^{O(M\sqrt{d}\log(Md/\varepsilon))}$ where $\|A-B\|_\infty \le \varepsilon$.*

6.5 Applications of Matrix-Vector Products

6.5.1 Preliminary Tools

We highlight specific prior results which we use in conjunction with our matrix-vector query upper bounds to obtain improved algorithmic results. First we recall a result of [29] which gives a nearly optimal low-rank approximation result in terms of the number of matrix-vector queries required.

Theorem 6.5.1 (Theorem 5.1 in [29])**.** *Given matrix-vector query access to a matrix $A \in \mathbb{R}^{n\times n}$, accuracy parameter $\varepsilon \in (0,1), k \in [n]$ and any $p \ge 1$, there exists an algorithm which uses $\tilde{O}(k/\varepsilon^{1/3})$ matrix-vector queries and outputs a $n \times k$ matrix Z with orthonormal columns such that with probability at least $9/10$,*

$$\|A(I-ZZ^T)\|_p \le (1+\varepsilon) \min_{U:U^TU=I_k} \|A(I-UU^T)\|_p$$

where $\|M\|_p = (\sum_{i=1}^n \sigma_i(M)^p)^{1/p}$ is the p-th Schatten norm where $\sigma_1, \ldots, \sigma(M)$ are the singular values of M. The runtime of the algorithm is $\tilde{O}(Tk/\varepsilon^{1/3} + nk^{w-1}/\varepsilon^{(w-1)/3})$ where T is the time for computing a matrix-vector query.

The second result is of [154] which give an optimized analysis of a variant of power method for computing the top k singular values.

Theorem 6.5.2 (Theorem 1 and 7 in [154])**.** *Given matrix-vector query access to a matrix $A \in \mathbb{R}^{n \times n}$, accuracy parameter $\varepsilon \in (0,1), k \in [n]$, there exists an algorithm which uses $\tilde{O}(k/\varepsilon^{1/2})$ matrix-vector queries and outputs a $1 \pm \varepsilon$ approximation to the top k singular values of A. The runtime of the algorithm is $\tilde{O}(Tk/\varepsilon^{1/2} + nk^2/\varepsilon + k^3/\varepsilon^{3/2})$.*

Lastly, we recall the guarantees of the classical conjugate-gradient descent method.

Theorem 6.5.3. *Let A be a symmetric PSD matrix and consider the linear system $Ax = b$ and let $x^* = \text{argmin}_x \|Ax - b\|_2$. Let κ denote the condition number of A. Given a starting vector x_0, the conjugate gradient descent algorithm uses $O(\sqrt{\kappa}\log(1/\varepsilon))$ matrix-vector queries and returns x such that*

$$\|x - x^*\|_A \le \varepsilon \|x_0 - x^*\|_A$$

where $\|x\|_A = (x^T A x)^{1/2}$ denotes the A-norm.

Note that matrices in our setting are also PSD, for example if $A_{i,j} = \langle x_i, x_j \rangle$. For non PSD matrices A, one can also use the conjugate gradient descent method on the matrix $A^T A$ which squares the condition number. Therefore, there are more complicated algorithms which work directly on the matrix-vector queries of A for non PSD matrices, for example see references in Chapters 6 and 7 of of [16]. We omit their discussion for clarity and just note that in practice, iterative methods which directly use matrix-vector queries are preferred for linear system solving.

6.5.2 Applications

We now derive specific applications using prior results from “matrix-free" methods. First we cover low-rank approximation.

For the ℓ_1 and ℓ_2^2 distance matrices, we improve upon prior works for computing a relative error low-rank approximation. While we can obtain such an approximation for a wide variety of Schatten norms, we state the bound in terms of the Frobenius norm since it has been studied in prior works.

Theorem 6.5.4. *Let $p \ge 1$ and consider the case that $A_{i,j} = \|x_i - x_j\|_p^p$ for all i, j. We can compute a matrix B such that*

$$\|A - B\|_F \le (1 + \varepsilon)\|A - A_k\|_F$$

where A_k denotes the optimal rank-k approximation to A in Frobenius norm. The runtime is $\tilde{O}(ndpk/\varepsilon^{1/3} + nk^{w-1}/\varepsilon^{(w-1)/3})$.

Proof. The theorem follows from combining the matrix-vector query runtime of Table 6.1 and Theorem 6.5.1. □

Note that the best prior result for the special case of ℓ_1 and ℓ_2^2 from [28] where they obtained a runtime bound of $O(ndk/\varepsilon + nk^{w-1}/\varepsilon^{w-1})$. Thus our bound improves upon this by a multiplicative factor of $\text{poly}(1/\varepsilon)$. We point out that the bound of $O(ndk/\varepsilon + nk^{w-1}/\varepsilon^{w-1})$ is actually *optimal* for the class of algorithms which sample the entries of A. Thus our results show that if we know the set of points beforehand, which is a natural assumption, one can overcome such lower bounds.

For the case of $A_{i,j} = \|x_i - x_j\|_2$, we cannot hope to achieve a relative error approximation for low-rank approximation since we only have fast matrix-vector queries to the matrix B where $B_{i,j} = (1 \pm \varepsilon)\|x_i - x_j\|_2$ via Theorem 6.2.11. Nevertheless, we can still obtain an additive error low-rank approximation guarantee which outperforms prior works. First we show that our approximation matrix-vector queries are sufficient to obtain such a guarantee.

Lemma 6.5.5. *Let A, B satisfy $\|A - B\|_F \le \varepsilon\|A\|_F$ and suppose A' and B' are the best rank-r approximation of A and B respectively in the Frobenius norm. Then*

$$\|A - B'\|_F \le \|A - A'\|_F + 2\varepsilon\|A\|_F.$$

Proof. We have

$$\begin{aligned}\|A - B'\|_F &\le \|A - B\|_F + \|B - B'\|_F \\ &\le \varepsilon\|A\|_F + \|B - A'\|_F \\ &\le \varepsilon\|A\|_F + \|B - A\|_F + \|A - A'\|_F \\ &\le \|A - A'\|_F + 2\varepsilon\|A\|_F.\end{aligned}$$

□

Theorem 6.5.6. *Let $A_{i,j} = \|x_i - x_j\|_2$ for all i, j. We can compute a matrix B such that*

$$\|A - B\|_F \le \|A - A_k\|_F + \varepsilon\|A\|_F$$

with probability $1 - 1/\text{poly}(n)$ where A_k denotes the optimal rank-k approximation to A in Frobenius norm. The runtime is $\tilde{O}(ndk/\varepsilon^{1/3} + nk^{w-1}/\varepsilon^{(w-1)/3})$.

Proof. The runtime follows from applying Theorem 6.5.1 on the matrix created after applying Theorems 2.0.2 and 6.2.11. The approximation guarantee follows from Lemma 6.5.5. □

The best prior work for additive error low-rank approximation for this case is due to [113] which obtained such a guarantee with runtime $\tilde{O}(nd \cdot \text{poly}(k, 1/\varepsilon))$ for a large unspecified polynomial in k and $1/\varepsilon$. Lastly we note that our relative error low-rank approximation guarantee holds for any f in Table 6.1, as summarized in Table 6.2.

Theorem 6.5.7. *Suppose we have exact matrix-vector query access to a matrix A with each query taking time T. Then we can output a matrix B such that*

$$\|A - B\|_F \le (1 + \varepsilon)\|A - A_k\|_F$$

where A_k denotes the optimal rank-k approximation to A in Frobenius norm. The runtime is $\tilde{O}(Tk/\varepsilon^{1/3} + nk^{w-1}/\varepsilon^{(w-1)/3})$.

Directly appealing to Theorems 6.5.2 and 6.5.3 in conjunction with our matrix-vector queries achieves the fastest runtime for computing the top k singular values and solving linear systems for a wide variety of distance matrices that we are aware of.

Theorem 6.5.8. *Suppose we have exact matrix-vector query access to a matrix A with each query taking time T. We can compute a $1 \pm \varepsilon$ approximation to the k singular values of A in time $\tilde{O}(T/\varepsilon^{1/2} + nk^2/\varepsilon + k^3/\varepsilon^{3/2})$. Furthermore, we can solve linear systems for A with the same guarantees as any iterative method which only uses matrix-vector queries with an multiplicative overhead of T.*

Finally, we can also perform matrix multiplication faster with distance matrices compared to the general runtime of $n^w \approx n^{2.37}$. This follows from the following lemma.

Lemma 6.5.9. *Suppose $A \in \mathbb{R}^{n\times n}$ admits an exact matrix-vector query algorithm in time T. Then for any $B \in \mathbb{R}^{n\times n}$, we can compute AB in time $O(Tm)$.*

Proof. We can compute AB by computing the product of A with the n columns of B separately. □

As a corollary, we obtain faster matrix multiplication for all the family of matrices which we have obtained a fast matrix-vector query for. We state one such corollary for the ℓ_p^p case.

Corollary 6.5.10. *Let $p \geq 1$ and consider the case that $A_{i,j} = \|x_i - x_j\|_p^p$ for all i, j. For any other matrix B, we can compute AB in time $O(n^2dp)$.*

We can improve upon the above result slightly if we are multiplying two distance matrices for the $p = 2$ case.

Lemma 6.5.11. *Consider the case that $A_{i,j} = \|x_i - x_j\|_2^2$ for all i, j and $B_{i,j} = \|y_i - y_j\|_2^2$, i.e., both A and B are $n \times n$ matrices with $f = \ell_2^2$. We can compute AB in time $O(n^2d^{w-2})$.*

Proof. By decomposing both A and B, it suffices to compute the product XX^TYY^T where $X, Y \in \mathbb{R}^{n\times d}$ are the matrices with the points x_i and y_i in the rows respectively. $Z_1 := X^TY \in \mathbb{R}^{d\times d}$ can be computed in $O(nd^2)$ time. Then $Z_2 := XZ_1 \in \mathbb{R}^{n\times d}$ can also be computed in $O(nd^2)$ time. Finally, we need to compute $Z_2 \times Y^T$. This can be done in $O(n^2d^{w-2})$ time by decomposing both Z_2 and Y^T into n/d many $d \times d$ square matrices and using the standard matrix multiplication bound on each pair of square matrices. This results in the claimed runtime of $O((n/d)^2 \cdot d^w) = O(n^2d^{w-2})$. □

6.6 A Fast Algorithm for Creating ℓ_1 and ℓ_2 Distance Matrices

We now present a fast algorithm for creating distance matrices which addresses our contribution (3) stated in the introduction. Given a set of n points $x_1, \ldots, x_n$ in $\mathbb{R}^d$, our goal is to initialize an approximate $n \times n$ distance matrix B for the ℓ_1 distance which satisfies

$$B_{ij} = (1 \pm \varepsilon)\|x_i - x_j\|_1 \tag{6.1}$$

for all entries of B where $0 < \varepsilon < 1$ is a precision parameter. The straightforward way to create the *exact* distance matrix takes take $O(n^2 d)$ time and by using the stability of Cauchy random variables, we can create B which satisfies (6.1) in $O(n^2 \log n)$ time for any constant ε. (Note the Johnson-Lindenstrauss lemma implies a similar guarantee for the ℓ_2 distance matrix). The goal of this section is to improve upon this 'baseline' runtime of $O(n^2 \log n)$. The final runtime guarantees of this section will be of the form $O(n^2 \cdot \text{poly}(\log \log n))$.

Our improvement holds in the **Word-RAM** model of computation. Formally, we assume each memory cell (i.e. word) can hold $O(\log n)$ bits and certain computations on words take $O(1)$ time. The only assumptions we require are the arithmetic operations of adding or subtracting words as well as performing left or right bit shifts on words takes constant time.

We first present prior work on metric compression of [111] in Section 6.6.1. Our algorithm description starts from Section 6.6.2 which describes our preprocesing step. Section 6.6.3 then presents our key algorithm ideas whose runtime and accuracy are analyzed in Sections 6.6.4 and 6.6.5.

6.6.1 Metric Compression Tree of [111]

The starting point of our result is the metric compression tree construction of [111], whose properties we summarize below. First we introduce some useful definitions. The aspect ration Φ of X is defined as

$$\Phi = \frac{\max_{i,j} \|x_i - x_j\|_1}{\min_{i \neq j} \|x_i - x_j\|_1}.$$

Let $\Delta' = \max_{i \in [n]} \|x_1 - x_i\|_1$ and $\Delta = 2^{\lceil \log \Delta' \rceil}$.

Theorem 1 of [111] implies the following result. Given a dataset $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ with aspect ration Φ, there exists a tree data structure T which allows for the computation of a compressed representation X for the purposes of distance computations. At a high level, T is created by enclosing X in a large enough and appropriately shifted axis-parallel square and then recursively dividing into smaller squares (also called cells) with half the side-length until all points of X are contained in their own cell. The edges of T encode the cell containment relationships. Formally, T has the following properties:

- The leaf nodes of T correspond to the points of X.
- The edges of T are of two types: short edges and long edges which are defined as follows. Short edges have a length d bit vector associated with them whereas long edges have an integer $\leq O(\log \Phi)$ associated with them.
- Each long edge with associated integer k represents a non-branching path of length k of short edges, all of whose associated length d bit vectors are the 0 string.
- Each node of T (including the nodes that are on paths which are compressed as long edges) have an associated level $-\infty < \ell \leq \log(4\Delta)$. A level ℓ of a node v corresponds to an axis-parallel square G_ℓ of side length 2^ℓ which contains all axis-parallel squares of child nodes of v.

The notion of a padded point is important for the metric compression properties of T.

Definition 6.6.1 (Padded Point)**.** *A point x_i is $(\varepsilon, \Lambda, \ell)$-padded, if the grid cell G_ℓ of side length 2^ℓ that contains x_i also contains the ball of radius $\rho(\ell)$ centered at x_i, where*

$$\rho(\ell) = 8\varepsilon^{-1}2^{\ell-\Lambda}\sqrt{d}.$$

We say that x_i is (ε, Λ)-padded in T, if it is $(\varepsilon, \Lambda, \ell)$-padded for every level ℓ.

The following lemma is proven in [111]. First define

$$\Lambda = \log(16d^{1.5}\log\Phi/(\varepsilon\delta)). \tag{6.2}$$

Lemma 6.6.1 (Lemma 1 in [111])**.** *Consider the construction of T defined formally in Section 3 of [111]. Every point x_i is (ε, Λ)-padded in T with probability $1 - \delta$.*

Now let x be any point in our dataset. We can construct $\widetilde{x} \in \mathbb{R}^d$, called the decompression of x, from T with the following procedure: We follow the downward path from the root of T to the leaf associated with x and collect a bit string for every coordinate d of $\widetilde{x}$. When going down a short edge with an associated bit vector b, we concatenate the ith bit of b to the end of the bit string that we are keeping track of for the ith coordinate of $\widetilde{x}$. When going down a long edge, we concatenate with a number of zeros equaling the integer associated with the long edge. Finally, a binary floating point is placed in the resulting bit strings of each coordinate after the bit corresponding to level 0. The collected bits then correspond to the binary expansion of the coordinates of $\widetilde{x}$. For a more thorough description of the decompression scheme, see Section 3 of [111].

The decompression scheme is useful because it allows approximate distance computations.

Lemma 6.6.2 (Lemma 2 in [111])**.** *If a point x_i is (ε, Λ)-padded in T, then for every $j \in [n]$,*

$$(1-\varepsilon)\|\widetilde{x}_i - \widetilde{x}_j\|_1 \le \|x_i - x_j\|_1 \le (1+\varepsilon)\|\widetilde{x}_i - \widetilde{x}_j\|_1.$$

We now cite the runtime and space required for T. The following theorem follows from the results of [111].

Theorem 6.6.3. *Let $L = \log\Phi + \Lambda$. T has $O(n\Lambda)$ edges, height L, its total size is $O(nd\Lambda + n\log n)$ bits, and its construction time is $O(ndL)$.*

We contrast the above guarantees with the naive representation of X which stores $O(\log n)$ bits of precision for each coordinate and occupies $O(nd\log n)$ bits of space, whereas T occupies roughly $O(nd\log\log n + n\log n)$ bits.

Finally, Theorem 2 in [111] implies we can create a collection of $O(\log n)$ trees T (by setting δ to be a small constant in (6.2)) such that every point in X is padded in at least one tree in the collection.

6.6.2 Step 1: Preprocessing Metric Compression Trees

We now describe the preprocessing steps needed for our faster distance matrix compression. Let

$$w = \frac{4d\Lambda}{\log n} \tag{6.3}$$

and recall our setting of Λ in (6.2). Note that we assume w is an integer which implicitly assumes $4d\Lambda \geq \log n$.

First we describe the preprocessing of T. Consider a short edge of T with an associated d length bit string b. We break up b into w equal chunks, each containing d/w bits. Consider a single chunk c. We pad (an equal number of) 2Λ many 0's after each bit in c so that the total number of bits is $\log n/2$. We then store each padded chunk in one word. We do this for every chunk resulting in w words for each short edge and we do this for all short edges in all the trees.

The second preprocessing step we perform is creating a $O(\sqrt{n}) \times O(\sqrt{n})$ table A. The rows and columns of A are indexed by all possible bit strings with $\frac{\log n}{2}$ bits. The entries of A record evaluations of the function $f(x,y)$ defined as follow: given x, y where $x, y \in \{0,1\}^{\frac{\log n}{2}}$, consider the partition of each of them into d/w blocks, each with an equal number of bits (2Λ bits per block. Note that $2\Lambda \cdot d/w = (\log n)/2$). Each block defines an integer. Doing so results in d/w integers $x^1, \ldots, x^{d/w}$ derived from x and w integers $y^1, \ldots, y^{d/w}$ derived from y. Finally,

$$f(x,y) = \sum_{i=1}^{d/w} |x^i - y^i|.$$

6.6.3 Step 2: Depth-First Search

We now calculate one row of the distance matrix from point a padded point x to all other points in our dataset. Our main subroutine is a tree search procedure. Its input is a node v in a tree T and it performs a depth-first search on the subtree rooted at v as described in Algorithm 18. Given an internal vertex v, it calculates all the distances between the padded point x to all data points in our dataset which are leaf nodes in the subtree rooted at v. A summary of the algorithm follows.

We perform a depth-first search starting at v. As we traverse the tree, we keep track of the current embedding of the internal nodes via collecting bits along the edges of T: we append bits when we descend the tree and remove as we move up edges. However, we only keep track of this embedding up to 2Λ levels below v. After that, we continue traversing the tree but don't update the embedding. The reason for this is after 2Λ levels, the embedding is precise enough for all nodes below with respect to computing the distance to x. Towards this end, we also track how many levels below v the tree search is currently at and update this value appropriately.

The current embedding is tracked using w words. Recall that the bit string of every short edge has been repackaged into w words, each 'responsible' for d/w coordinates. Furthermore, in each word on the edge, we have padded 0's between the bits of each d/w coordinates. When we need to update the current embedding by incorporating the bits along a short edge e, we simply perform a bit shift on each of the w words on e and add it to the w words we are keeping track of. We need to make sure we place the bits 'in order.' That is for our tracked embedding, for every d coordinates, the bits on an edge e should precede the bits on the edge directly following e in the tree search. Due to the padding from the preprocessing step, the bit shift implies the bits on the short edges after e will be placed in their appropriate corresponding places in order in the embedding representation.

Algorithm 18 DFS in Subtree

```
1: Input: Metric Compression Tree $T$, node $v$
2: procedure SEARCH( $T, v$)
3:     Initialize a global counter $p$ for the number of levels which have been processed. Initially set to 0 and will be at most $2\Lambda$
4:     Initialize $w$ words $t_1, \ldots, t_w$, all initially 0
5:     Initialize a global counter $r$ for the current level which is initially set to the level of $v$ in $T$
6:     Perform a depth-first search in the subtree rooted at $v$. Run Process-Short-Edge if a short edge is encountered, Process-Long-Edge if a long edge is encountered, and Process-Leaf when we arrive at a leaf.
7: end procedure
```

While performing the depth-first search, we will encounter both short and long edges. When encountering a short edge, we run the function `Process-Short-Edge` and similarly, we run `Process-Long-Edge` when a long edge is encountered. Finally if we arrive at a leaf node, we run `Process-Leaf`.

Algorithm 19 Process Short Edge

```
1: Input: Short edge $e$, number of processed nodes $p$, $t_1, \ldots, t_w$
2: procedure PROCESS-SHORT-EDGE($e, p, t_1, \ldots, t_w$)
3:     Let $e_1, \ldots, e_w$ be the $w$ words associated with edge $e$
4:     If search is traversing down $e$ and $p < 2\Lambda$, add $2^{-p} e_i$ to $t_i$ for all $1 \leq i \leq w$ and increment $p$
5:     If search is traversing up $e$ and $p \leq 2\Lambda$, subtract $2^{-p} e_i$ from $t_i$ for all $1 \leq i \leq w$ and decrement $p$
6:     Update $r$ to the level of the current node
7: end procedure
```

Algorithm 20 Process Long Edge

```
1: Input: Long edge $e$, number of processed nodes $p$
2: procedure PROCESS-LONG-EDGE($e, p$)
3:     If search is traversing down $e$ and $p < 2\Lambda$, increment $p$
4:     If search is traversing up $e$ and $p \leq 2\Lambda$, decrement $p$
5:     Update $r$ to the level of the current node
6: end procedure
```

When we arrive at a leaf node y, we currently have the decompression of y computed from the tree. Note that we only have kept track of the bits after node v (up to limited precision) since all prior bits are the same for y and x since they are in the same subtree. More specifically, we have w words $t_1, \dots, t_w$. The first word t_1 has 2Λ bits of each of the first d/w coordinates of y. For every coordinate, the 2Λ bits respect the order described in the decompression step in Section 6.6.2. A similar fact is true for the rest of the words t_i. Now to calculate the distance between x and y, we just have to consider the 2Λ bits of all d coordinates of x which come after descending down vertex v. We then repackage these $2d\Lambda$ total bits into w words in the same format as y. Note this preprocessing for x only happens once (at the subtreee level) and can be used for all leaves in the subtree rooted at v.

Algorithm 21 Process Leaf

```
1: Input: $t_1, \dots, t_w$
2: procedure PROCESS-LEAF($y, t_1, \dots, t_w$)
3:     Let the point $y$ correspond to the current leaf node
4:     Let $s_1, \dots, s_w$ denote the embedding of $x$ after node $v$, preprocessed to be in the same
   format as $t_1, \dots, t_w$
5:     Report $\sum_{i=1}^{w} A[t_i, s_i]$ as the distance between $x$ and $y$
6: end procedure
```

Finally, the complete algorithm just calls Algorithm 18 on successive parent nodes of x. We mark each subtree that has already been processed (at the root node) so that the subtree is only ever visited once. The number of calls to Algorithm 18 is at most the height of the tree, which is bounded by $O(\log \Phi + \Lambda)$. We then repeat this for all points x in our dataset (using the tree which x is padded in) to create the full distance matrix.

6.6.4 Runtime Analysis

We consider the runtime required to compute the row corresponding to a padded point x in the distance matrix. Multiplying by n results in the total runtime. Consider the tree T in which x is padded in and which we use for the algorithm described in the previous section and recall the properties of T outlined in Theorem 6.6.3. T has $O(n\Lambda)$ edges, each of which is only visited at most twice in the tree search (going up and down). Thus the time to traverse the tree is $O(n\Lambda)$. There are also at most $O(n\Lambda)$ short edges in T. Updating the embedding given by $t_1, \dots, t_w$ takes $O(w)$ time per edge since it can be done in $O(w)$ total word operations. Long edges don't require this time since they represent 0 bits; for long edges, we just increment the counter for the current level. Altogether, the total runtime for updating $t_1, \dots, t_w$ across all calls to Algorithm 18 for the padded point x is $O(n\Lambda w)$. Finally, calculating the distance from x to a fixed point y requires $O(w)$ time since we just index into the array A w times. Thus the total runtime is dominated by $O(n\Lambda w)$. Finally, the total runtime for computing all rows of the distance matrix is

$$O(n^2\Lambda w) = O\left(\frac{n^2 d\Lambda^2}{\log n}\right) = O\left(\frac{n^2 d}{\log n}\log^2\left(\frac{d\log\Phi}{\varepsilon}\right)\right)$$

by setting δ to be a small constant in (6.2).

6.6.5 Accuracy Analysis

We now show that the distances we calculate are accurate within a $1 \pm \varepsilon$ multiplicative factor. The lemma below shows that if a padded point x and another point y have a sufficiently far away least-common ancestor in T, then we can disregard many lower order bits in the decompression computed from T while still guaranteeing accurate distance measurements. The lemma crucially relies on x being padded.

Lemma 6.6.4. *Suppose x is (ε, Λ)-padded in T. For another point y, suppose the least common ancestor of x and y is at level ℓ. Let $\widetilde{x}$ and $\widetilde{y}$ denote the sketches of x and y produced by T. Let $\widetilde{x}'$ be a modified version of $\widetilde{x}$ where for each of the d coordinates, we remove all the bits acquired after level $\ell - 2\Lambda$. Similarly define $\widetilde{y}'$. Then*

$$\|\widetilde{x}' - \widetilde{y}'\|_1 = (1 \pm \varepsilon)\|x - y\|_1.$$

Proof. Since x is padded, we know that $\|x-y\|_1 \ge p(\ell - 1)$ by Definition 6.6.1. On the other hand, if we ignore the bits after level $\ell - 2\Lambda$ for every coordinate of $\widetilde{x}$ and $\widetilde{y}$, the additive approximation error in the distance is bounded by a constant factor times

$$d \cdot \sum_{i=-\infty}^{\ell-2\Lambda-1} 2^i = d \cdot 2^{\ell - 2\Lambda}.$$

From our choice of Λ, we can easily verify that $d \cdot 2^{\ell-2\Lambda} \le \varepsilon p(\ell-1)/2$. Putting everything together and adjusting the value of ε, we have

$$\|\widetilde{x}' - \widetilde{y}'\|_1 = \|\widetilde{x} - \widetilde{y}\|_1 \pm \varepsilon p(\ell-1)/2 = (1 \pm \varepsilon/2)\|x-y\|_1 \pm \varepsilon p(\ell-1)/2 = (1\pm\varepsilon)\|x-y\|_1$$

where we have used the fact that $\|\widetilde{x} - \widetilde{y}\|_1 = (1 \pm \varepsilon/2)\|x-y\|_1$ from the guarantees of the compression tree of [111]. □

Putting together our results above along with the Johnson-Lindenstrauss Lemma and Theorem 2.0.2 proves the following theorem.

Theorem 6.6.5. *Let $X = \{x_1, \dots, x_n\}$ be a dataset of n points in d dimensions with aspect ration Φ. We can calculate a $n \times n$ matrix B such that each (i, j) entry B_{ij} of B satisfies*

$$(1-\varepsilon)\|x_i - x_j\|_1 \leq B_{ij} \leq (1+\varepsilon)\|x_i - x_j\|_1$$

in time

$$O\left(\frac{n^2 d}{\log n} \log^2\left(\frac{d \log \Phi}{\varepsilon}\right)\right).$$

Assuming the aspect ratio is polynomially bounded, we can compute $n \times n$ matrix B such that each (i, j) entry B_{ij} of B satisfies

$$(1-\varepsilon)\|x_i - x_j\|_2 \leq B_{ij} \leq (1+\varepsilon)\|x_i - x_j\|_2$$

with probability $1 - 1/poly(n)$. The construction time is

$$O\left(\frac{n^2}{\varepsilon^2} \log^2\left(\frac{\log n}{\varepsilon}\right)\right).$$

6.6.6 A Faster Algorithm for ℓ_∞ Distance Matrix Construction Over Bounded Alphabet

In this section, we show how to create the ℓ_∞ distance matrix. Recall from Section 6.4 that there exists no $o(n^2)$ time algorithm to compute a matrix-vector query for the ℓ_∞ distance matrix, assuming SETH, even for n points in $\{0, 1, 2\}^d$. This suggests that any algorithm for computing a matrix-vector query needs to initialize the distance matrix. However, there is still a gap between a $\Omega(n^2)$ lower bound for matrix-vector queries and the naive $O(n^2 d)$ time needed to compute the ℓ_∞ distance matrix. We make progress towards showing that this gap is not necessary. Our main result is that surprisingly, we can *initialize* the ℓ_∞ distance matrix in time much faster than the naive $O(n^2 d)$ time.

Theorem 6.6.6. *Given n points over $\{0, 1, \dots, M\}^d$, we can initialize the exact ℓ_∞ distance matrix in time $O(M^{w-1} n^2 (d \log M)^{w-2})$ where w is the matrix multiplication constant. We can also initialize a $n \times n$ matrix B such that each (i, j) entry B_{ij} of B satisfies*

$$(1-\varepsilon)\|x_i - x_j\|_\infty \leq B_{ij} \leq (1+\varepsilon)\|x_i - x_j\|_\infty$$

in time $\tilde{O}(\varepsilon^{-1} n^2 (dM)^{w-2})$.

Thus for $M = O(1)$, which is the setting of the lower bound, we can initialize the distance matrix in time $O(n^2 d^{w-2})$ and thus compute a matrix-vector query in that time as well.

Proof. The starting point of the proof is to first design an algorithm which constructs a matrix with (i, j) entry an indicator vector for $\|x - y\|_\infty \le i$ or $\|x - y\|_\infty > i$. Given this, we can then sum across all M choices and construct the full distance matrix. Thus it suffices to solve this intermediate task.

Pick a sufficiently large p such that $di^p \le (i+1)^p$. A choice of $p = O(M \log d)$ suffices. Now in the case that $\|x-y\|_\infty \le i$, we have $\|x-y\|_p^p \le di^p$ and otherwise, $\|x-y\|_p^p \ge (i+1)^p$. Thus, the matrix C with the (i, j) entry being $\|x_i - x_j\|_p^p$ is able to distinguish the two cases so it suffices to create such a matrix. Now we can write $\|x - y\|_p^p$ as an inner product in $O(pd)$ variables, i.e., it is a gram matrix. Thus computing C can be done by computing a product of $n \times O(pd)$ matrix by a $O(pd) \times n$ matrix, which can be done in

$$O\left(\left(\frac{n}{pd}\right)^2 (pd)^{w-2}\right) = O(n^2 (pd)^{w-2}).$$

time by partitioning each matrix into square submatrices of dimension $O(pd)$. Plugging in the bound for p and considering all possible choices of i results in the final runtime bound of $O(M^{w-1} n^2 (d \log M)^{w-2})$, as desired.

Now if we only want to approximate each entry up to a multiplicative $1 \pm \varepsilon$ factor, it suffices to only loop over i's which are increasing by powers of $1 + c\varepsilon$ for a small constant c. This replaces an $O(M)$ factor by an $O(\varepsilon^{-1} \log M)$ factor. $\square$

6.7 Empirical Evaluation

We perform an empirical evaluation of our matrix-vector query for the ℓ_1 distance function. We chose to implement our ℓ_1 upper bound since it's a clean algorithm which possesses many of the same underlying algorithmic ideas as some of our other upper bound results. We envision that similar empirical results hold for most of our upper bounds in Table 6.1. Furthermore, matrix-vector queries are the dominating subroutine in many key practical linear algebra algorithms such as the power method for eigenvalue estimation or iterative methods for linear regression: a fast matrix-vector query runtime automatically translates to faster algorithms for downstream applications.

Experimental Design. We chose two real and one synthetic dataset for our experiments. We have two “small" datasets and one “large" dataset. The two small datasets have $5 \cdot 10^4$ points whereas the large dataset has approximately 10^6 points. The first dataset is points drawn from a mixture of three spherical Gaussians in $\mathbb{R}^{50}$. The second dataset is the standard MNIST dataset [132] and finally, our large dataset is Glove word embeddings[1] in $\mathbb{R}^{50}$ [158].

The two small datasets are small enough that one can feasibly initialize the full $n \times n$ distance matrix in memory in reasonable time. A $5 \cdot 10^4 \times 5 \cdot 10^4$ matrix with each entry stored using 32 bits requires 10 gigabytes of space. This is simply impossible for the Glove

[1]Can be accessed here: http://github.com/erikbern/ann-benchmarks/

Dataset	(n, d)	Algo.	Preprocessing Time	Avg. Query Time
Gaussian Mixture	$(5 \cdot 10^4, 50)$	Naive	453.7 s	43.3 s
		Ours	0.55 s	0.09 s
MNIST	$(5 \cdot 10^4, 784)$	Naive	2672.5 s	38.6 s
		Ours	5.5 s	1.9 s
Glove	$(1.2 \cdot 10^6, 50)$	Naive	-	$\approx$ 2.6 days (estimated)
		Ours	16.8 s	3.4 s

Table 6.3: Dataset description and empirical results. (n, d) denotes the number of points and dimension of the dataset, respectively. Query times are averaged over 10 trials with Gaussian vectors as queries.

dataset as approximately 5.8 terabytes of space is required to initialize the distance matrix (in contrast, the dataset itself only requires < 0.3 gigabytes to store).

The naive algorithm for the small datasets is the following: we initialize the full distance matrix (which will count towards preprocessing), and then we use the full distance matrix to perform a matrix-vector query. Note that having the full matrix to perform a matrix-vector product only helps the naive algorithm since it can now take advantage of optimized linear algebra subroutines for matrix multiplication and does not need to explicitly calculate the matrix entries. Since we cannot initialize the full distance matrix for the large dataset, the naive algorithm in this case will compute the matrix-vector product in a standalone fashion by generating the entries of the distance matrix on the fly. We compare the naive algorithm to our Algorithms 8 and 9.

Our experiments are done in a 2021 M1 Macbook Pro with 32 gigabytes of RAM. We implement all algorithms in Python 3.9 using Numpy with Numba acceleration to speed up all algorithms whenever possible.

Results. Results are shown in Table 6.3. We show preprocessing and query time for both the naive and our algorithm in seconds. The query time is averaged over 10 trials using Gaussian vectors as queries. For the Glove dataset, it was infeasible to calculate even a single matrix-vector product, even using fast Numba accelerated code. We thus estimated the full query time by calculating the time on a subset of $5 \cdot 10^4$ points of the Glove dataset and extrapolating to the full dataset by multiplying the query time by $(n/(5 \cdot 10^4))^2$ where n is the total number of points. We see that in all cases, our algorithm outperforms the naive algorithm in both preprocessing time and query time and the gains become increasingly substantial as the dataset size increases, as predicted by our theoretical results.

Chapter 7

Subquadratic Algorithms for Kernel Matrices

For a kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and a set $X = \{x_1 \dots x_n\} \subset \mathbb{R}^d$ of n points, the entries of the $n \times n$ kernel matrix K are defined as $K_{i,j} = k(x_i, x_j)$. Alternatively, one can view X as the vertex set of a complete weighted graph where the weights between points are defined by the kernel matrix K. Popular choices of kernel functions k include the Gaussian kernel, the Laplace kernel, exponential kernel, etc. The following notion of a *kernel density estimation* query is central to the chapter.

Definition 1.6.1. (Kernel Density Estimation (KDE) Queries) *For a given dataset $X \subset \mathbb{R}^d$ of size n, kernel function k, and precision parameter $\varepsilon > 0$, a KDE data structure supports the following operation: given a query $y \in \mathbb{R}^d$, return a value $\mathrm{KDE}_X(y)$ that lies in the interval $[(1-\varepsilon)z, (1+\varepsilon)\, z]$, where $z = \sum_{x \in X} k(x, y)$, assuming that $k(x, y) \geq \tau$ for all $x \in X$.*

The performance of the state of the art algorithms for KDE also scales proportional to the smallest kernel value of the dataset (see Table 2.1). In short, after a preprocessing time that is sub-quadratic (in n), KDE data structures use time sublinear in n to answer queries defined as above. Note that for all of our kernels, $k(x, y) \leq 1$ for all inputs x, y.

In this chapter, we show that given a KDE data structure as described above, it is possible to solve a variety of matrix and graph problems in time subquadratic time $o(n^2)$, i.e., sublinear in the matrix size. We emphasize that in our applications, we only require *black-box* access to KDE queries. Given this, we design such algorithms for problems such as eigenvalue/eigenvector estimation, low-rank approximation, and graph sparsification.

Our recall that our results are obtained via the following two-pronged approach. First, we use KDE data structures to design algorithms for the following basic primitives, frequently used in sublinear time algorithms and property testing:

1. sampling vertices by their (weighted) degree in K (Theorems 7.3.2 and 7.3.4 and Algorithms 23 / 25),

2. sampling random neighbors of a given vertex by edge weights in K and sampling a random weighted edge (Theorem 7.3.5 and Algorithms 26 and 27),

3. performing random walks in the graph K (Theorem 7.3.7 and Algorithm 28), and

4. sampling the rows of the edge-vertex incident matrix and the kernel matrix K, both with probability proportional to respective row norms squared (Section 7.4.1, Theorem 7.4.1, and Section 7.4.2, Corollary 7.4.10 respectively).

In the second step, we use these primitives to implement a host of algorithms for the aforementioned problems. We emphasize that these primitives are used in a black-box manner, meaning that any further improvements to their running times will automatically translate into improved algorithms for the downstream problems. For our applications, we make the following parameterization, which we expand upon in Remark 7.2.1 and Section 7.2.1. At a high level, many of our applications, such as spectral sparsification, are succinctly characterized by the following parameterization. Recall parameterization 1.6.1: All of our algorithms are parameterized by the smallest edge weight in the kernel matrix, i.e., the smallest edge weight in the matrix K is at least τ.

Table 1.1 lists our applications along with the number of KDE queries required in addition to any post-processing time. We refer to the specific sections of the body listed below for full details. We note that in *all of our theorems below*, we assume access to a KDE data structure of Definition 1.6.1 with parameters ε and τ.

One of our main results is spectral sparsification of the kernel matrix K interpreted as a weighted graph. In Section 7.4.1, we compute a sparse subgraph whose associated matrix closely approximates that of the kernel matrix K. The most meaningful matrix to study for such a sparsification is the Laplacian matrix, defined as $D-K$ where D is a diagonal matrix of vertex degrees. The Laplacian matrix encodes fundamental combinatorial properties of the underlying graph and has been well-studied for numerous applications, including sparsification; see [32, 149, 181] for a survey of the Laplacian and its applications. Our result computes a sparse graph, with a number of edges that is linear in n, whose Laplacian matrix spectrally approximates the Laplacian matrix of the original graph K under Parameterization 1.6.1.

Theorem 7.0.1 (Informal; see Thm. 7.4.1)**.** *Let L be the Laplacian matrix corresponding to the graph K. Then, for any $\varepsilon \in (0,1)$, there exists an algorithm that outputs a weighted graph G' with only $m = O(n\log n/(\varepsilon^2\tau^3))$ edges, such that with probability at least $9/10$, $(1-\varepsilon)L \preceq L_{G'} \preceq (1+\varepsilon)L$. The algorithm makes $\widetilde{O}(m/\tau^3)$ KDE queries and requires $\widetilde{O}(md/\tau^3)$ post-processing time.*

We compare our results with prior works in Remark 7.2.1. We also show that Parameterization 1.6.1 is **inherent** for spectral sparsification. In particular, we use a hardness result from [10] to show that for the Gaussian kernel, under the strong exponential time hypothesis [103], any algorithm that returns an $O(1)$-approximate spectral sparsifier with $O(n^{1.99})$ edges requires $\Omega\left(n \cdot 2^{\log(1/\tau)^{0.32}}\right)$ time (see Theorem 7.4.4 for a formal statement). Obtaining the optimal dependence on τ remains an outstanding open question, even for

Gaussian and Laplace kernels. Spectral sparsification has further downstream applications in solving Laplacian linear systems, which we present in Section 7.4.1.

Furthermore, obtain truly sublinear time algorithms for approximating the top eigenvalue and eigenvector of the kernel matrix, a problem which was studied in [24]. Our result is the following theorem. Our bounds, and those of prior work, depend on the parameter p, which refers to the exponent of τ in the KDE query runtimes. For example for the Gaussian kernel, $p \approx 0.173$. See Table 2.1 for other kernels.

Theorem 7.0.2 (Informal; see Theorem 7.4.14)**.** *Given an $n \times n$ kernel matrix K that admits a KDE data-structure with query time $d/(\varepsilon^2\tau^p)$ (Table 2.1), there exists an algorithm that outputs a unit vector v such that $v^T K v \geq (1-\varepsilon)\lambda_1(K)$ in time $\min\Big(\tilde{O}(d/(\varepsilon^{4.5}\tau^4)), \tilde{O}(d/(\varepsilon^{9+6p}\tau^{2+2p}))\Big)$, where $\lambda_1(K)$ denotes the largest eigenvalue of K.*

We discuss related works in Remark 7.2.2. In summary, the best prior result of [24] had a runtime of $\Omega(n^{1+p})$ whereas our bound has no dependence on n. Finally, our last linear-algebraic result is an additive-error low-rank approximation of the kernel matrix, presented in Section 7.4.2.

Theorem 7.0.3 (Informal; see Cor. 7.4.10)**.** *There exists an algorithm that outputs a rank r matrix B such that $\|K - B\|_F^2 \leq \|K - K_r\|_F^2 + \varepsilon\|K\|_F^2$ with probability 99% where K_r is the optimal rank-r approximation of K. It uses n KDE queries and $O(n \cdot \mathrm{poly}(r, 1/\varepsilon) + nrd/\varepsilon)$ post-processing time.*

We give detailed comparisons between our results and prior work in Remark 7.2.3. As a summary, [28] obtain a *relative* error approximation with a running time of $\widetilde{O}\left(nd\,(r/\varepsilon)^{\omega-1}\right)$, where ω denotes the matrix multiplication constant, whereas our running time is dominated by $O(nrd/\varepsilon)$ and we obtain only additive error guarantees. Nevertheless, the algorithm we obtain, which builds upon the sampling scheme of [87], is a conceptually simpler algorithm than the algorithm of [28] and easier to empirically evaluate. Indeed, we implement this algorithm in Section 7.5 and show that it is highly competitive to the SVD.

We note that there is a line of work that considers dimensionality reduction for kernel density estimation e.g., through coresets [160, 161, 187]. We view this direction of work as orthogonal to our line of study. Lastly, the work [24] is similar in spirit to our work as they also utilize KDE queries to speed up algorithms for kernel matrices. Besides top eigenvalue estimation mentioned before, [24] also study the problem of estimating the sum of all entries in the kernel matrix and obtain tight bounds for the latter.

7.1 Technical Overview

We provide a high-level overview and intuition for our algorithms. We first highlight our algorithmic building blocks for fundamental tasks and then describe how these components can be used to handle a wide range of problems. We note that our building blocks use KDE data structures in a black-box way and thus we describe their performance in terms of the number of queries to a KDE oracle. We also note that a permeating theme across all subsequent applications is that we want to perform some algorithmic task on a kernel matrix K without computing each of its entries $k(x_i, x_j)$.

Algorithmic Building Blocks . We first describe the "multi-level" KDE data structure, which constructs a KDE data structure on the entire input dataset X, and then recursively partitions X into two halves, building a KDE data structure on each half. The main observation here is that if the initialization of a KDE data structure uses runtime linear in the size n of X, then at each recursive level, the initialization of the KDE data structures across all partitions remains linear. Since there are $O(\log n)$ levels, the overall runtime to initialize our multi-level KDE data structure incurs only a logarithmic overhead (see Figure 7.1 for an illustration).

Weighted vertex sampling. We describe how to sample vertices approximately proportional to their weighted degree, where the weighted degree of a vertex x_i with $i \in [n]$ is $w_i = \sum_{j \neq i} k(x_i, x_j)$. We observe that performing n KDE queries suffices to get an approximation of the weighted vertex degree of all n vertices. We can thus think of vertex sampling as a preprocessing step that uses n queries upfront and then allows for arbitrary sample access at any point in the future with no query cost. Moreover, this preprocessing step of taking n queries only needs to be performed once. Further, we can then perform weighted vertex sampling from a distribution that is ε-close in total variation to the true distribution (see Theorem 7.3.4 for details). Here, we use a multi-level tree structure to iteratively choose a subset of vertices with probability proportional to its approximate sum of weighted degrees determined by the preprocessing step, until the final vertex is sampled. Hence after the initial n KDE queries, each query only uses $O(\log n)$ runtime, which is significantly better than the naïve implementation that uses quadratic time to compute the entire kernel matrix.

Weighted neighbor edge sampling. We describe how to perform weighted neighbor edge sampling for a given vertex x. The goal of weighted neighbor edge sampling is to efficiently output a vertex v such that $\Pr[v = x_k] = \frac{(1 \pm \varepsilon) k(x, x_k)}{\sum_{j \in [n], x_j \neq x} k(x, x_j)}$ for all $k \in [n]$. Unlike the degree case, edge sampling is not a straightforward KDE query since the sampling probability is proportional to the kernel value between two points, rather than the sum of multiple kernel values that a KDE query provides. However, we can utilize a similar tree procedure as in Figure 7.1 in conjunction with KDE queries.

In particular, consider the tree in Figure 7.1 where each internal node corresponds to a subset of neighbors of x. The two children of a parent node in the tree are simply the two approximately equal subsets whose union make up the subset representing the parent node. We can descend down the tree using the same probabilistic procedure as in the vertex sampling case: at every node, we pick one of the children to descend into with probability proportional to the sum of the edge weights represented by the children. The sum of edge weights of the children can be approximated by a query to an appropriate KDE data structure in the “multi-level" KDE data structure described previously. By appropriately decreasing the error of KDE data structures at each level of the tree, the sampled neighbor satisfies the aforementioned sampling guarantee. Since the tree has height $O(\log n)$, then we can perform weighted neighbor edge sampling, up to a tunably small total variation distance, using $O(\log n)$ KDE queries and $O(\log n)$ time (see theorems 7.3.5 and 7.3.6 for details).

Random walks. We use our edge sampling procedure to output a random walk on the kernel graph, where at any current vertex v of the walk, the next neighbor of v visited by the random walk is chosen with probability proportional to the edge weights adjacent to v. In particular, for a random walk with T steps, we can simply sequentially call our edge sampling procedure T times, with each instance corresponding to a separate step in the random walk. Thus we can perform T steps of a random walk, again up to a tunably small total variation distance, using $O(T \log n)$ KDE queries and $O(T \log n)$ additional time.

Importance Sampling for the edge-vertex incidence matrix and the kernel matrix. We now describe how to sample the rows of the edge vertex incident matrix H and the kernel matrix K with probability proportional to the importance sampling score / leverage score (see Definition 7.4.2). We remark that approximately sampling proportional to the *leverage score* distribution for H is a fundamental algorithmic primitive in spectral graph theory and numerical linear algebra. We note that a priori, such a task seems impossible to perform in $o(n^2)$ time, even if the leverage scores are precomputed for us, since the support of the distribution has size $\Theta(n^2)$. However, note we do not need to compute (even approximately) each leverage score to perform the sampling, but rather just output an edge proportional to the right distribution.

We accomplish this by instead sampling proportional to the squared Euclidean norm of the rows of H. It is known that oversampling the rows of a matrix by a factor that depends on the condition number is sufficient to approximate leverage score sampling (see proof of Theorem 7.4.1). Further, we show that H has a condition number (Lemma 7.4.3) that is bounded by $\text{poly}(1/\tau)$. Recall, the edge-vertex incident matrix is defined as the $\binom{n}{2} \times n$ matrix with the rows indexed by all possible edges and the columns indexed by vertices. For each $e = \{i, j\}$, we have $H_{\{i,j\},i} = \sqrt{k(x_i, x_j)}$ and $H_{\{i,j\},j} = -\sqrt{k(x_i, x_j)}$. We pick the ordering of i and j arbitrarily. Note that this is a weighted analogue of the standard edge-vertex incident matrix and satisfies $H^T H = L_G$ where L_G is the Laplacian matrix of the graph corresponding to the kernel matrix K. For both H and K, we wish to sample the rows with probability proportional to row normed squared. For example, the row r_e corresponding to edge $e = (x_i, x_j)$ in H satisfies $\|r_e\|_2^2 = 2k(x_i, x_j)$. Since the squared norm of each row is proportional to the weight of the corresponding edge, we can perform this sampling by combining the weighted vertex sampling and weighted neighbor edge sampling primitives: we first sample a vertex with probability proportional to its degree and then sample an appropriate random neighbor. Thus our row norm sampling procedure is sufficient to simulate leverage score sampling (up to a condition number factor), which implies our downstream application of spectral sparsification.

We now describe the related primitive of sampling the rows of the kernel matrix K. Naïvely performing this sampling would require us to implicitly compute the entire kernel matrix, which as mentioned previously, is prohibitive. However, if there exists a constant c such that the kernel function k that defines the matrix K satisfies $k(x,y)^2 = k(cx,cy)$ for all inputs x,y, then the ℓ_2^2 norm of each row can be approximated via a KDE query on the transformed dataset $X' = cX$. In particular, the ℓ_2^2 row norms of K are the vertex degrees of the kernel graph for X'. The property that $k(x,y)^2 = k(cx,cy)$ holds for the most popular kernels such as the Laplacian, exponential, and Gaussian kernels. Thus, we can sample the rows of the kernel matrix with the desired probabilities.

7.1.1 Linear Algebra Applications

We now discuss our linear algebra applications.

Spectral sparsification. Using the previously described primitives of weighted vertex sampling and weighted neighbor edge sampling, we show that an ε spectral sparsifier for the kernel density graph G can be computed i.e., we compute a graph G' such that for all vectors x, $(1-\varepsilon)x^T L_G x \le x^T L_{G'} x \le (1+\varepsilon)x^T L_{G'} x$, where L_G and $L_{G'}$ denote the Laplacian matrices of the graphs G and G'. Recall that H is the $\binom{n}{2} \times n$ matrix such that $H_{\{i,j\},i} = \sqrt{k(x_i,x_j)}$ and $H_{\{i,j\},j} = -\sqrt{k(x_i,x_j)}$. Here we use subsets of $[n]$ of size 2 to index the rows of H and the entry to be made negative in the above definition is picked arbitrarily. It can be verified that $H^T H = L_G$. It is known that sampling $t = O(n\log(n)/\varepsilon^2)$ rows of the matrix H by using the so-called leverage scores gives a $t \times \binom{n}{2}$ selecting-and-scaling matrix S such that with probability at least 9/10,

$$(1-\varepsilon)L_G = (1-\varepsilon)H^T H \preceq H^T S^T S H \preceq (1+\varepsilon)H^T H = (1+\varepsilon)L_G. \tag{7.1}$$

Thus the matrix SH directly corresponds to a graph G', which is an ε spectral sparsifier for graph G. The leverage scores of rows of H are also called "effective resistances" of edges of graph G. Unfortunately, with the edge and neighbor vertex sampling primitives that we have, we cannot perform leverage score sampling of H. On the other hand, observe that the squared norm of row $\{i,j\}$ of H is $2k(x_i,x_j)$ and with an application of vertex sampling and edge sampling, we can sample a row of H from the length squared distribution i.e., the distribution on rows where probability of sampling a row is proportional to its squared norm. It is a standard result that sampling from squared length distribution gives a selecting-and-scaling matrix S that satisfies (7.1), although we have to sample $t = O(\kappa^2 n\log(n)/\varepsilon^2)$ rows from this distribution, where $\kappa = \sigma_{\max}(H)/\sigma_{\min}(H)$ denotes the condition number of H ($\sigma_{\max}(H)/\sigma_{\min}(H)$ denote the largest/smallest *positive* singular values).

With the parameterization that for all $i \ne j$, $k(x_i,x_j) \ge \tau$, we are able to show that $\kappa \le O(1/\tau^{1.5})$. Importantly, our upper bound on the condition number is independent of the data dimension and number of input points. We obtain the upper bound on condition number by using a Cheeger-type inequality for weighted graphs. Note that $\sigma_{\min}(H) \ge \sqrt{\lambda_2(H^T H)} = \sqrt{\lambda_2(L_G)}$, where we use $\lambda_2(M)$ to denote the second smallest eigenvalue of a positive semidefinite matrix. Cheeger's inequality lower bounds exactly the quantity $\lambda_2(L_G)$ in terms of graph conductance. A lower bound of τ on every kernel value implies that every node in the Kernel Graph has a high weighted degree and this lets us lower bound $\lambda_2(G)$ in terms of τ using a Cheeger-type inequality from [86] and shows that $O(n\log(n)/\tau^3\varepsilon^2)$ samples from the approximate squared length sampling distribution gives an ε spectral sparsifier for the graph G.

First eigenvalue and eigenvector approximation. Our goal is to compute a $1-\varepsilon$ approximation to λ, the first eigenvalue of K, and an accompanying approximate eigenvector. Such a task is key in kernel PCA and related methods. We begin by noting that under the natural constraint that each row of K sums to at least $n\tau$, a condition used in prior works [24], the first eigenvalue must be at least $n\tau$ by looking at the quadratic form associated with the all-ones vector.

Now we combine two disparate families of algorithms: first the guarantees of [27, 36] show that sub-sampling a $t \times t$ principal submatrix of a PSD matrix preserves the eigenvalues of the matrix up to an additive $O(n/\sqrt{t})$ factor. Since we've shown the first eigenvalue of K is at least $n\tau$, we can set t roughly $O(1/(\varepsilon^2\tau^2))$ with the guarantee that the top eigenvalue of the sub-sampled matrix is at lest $(1-\varepsilon)\lambda$. Now we can either run the standard Krylov method algorithm [153] to compute the top eigenvalue of the sampled matrix or alternatively, we can instead use the algorithm of [24], the prior state of the art, to compute the eigenvalues of the sampled matrix. At a high level, their algorithm utilizes KDE queries to approximately perform power method on the kernel graph without creating the kernel matrix. In our case, we can instead run their algorithm on the smaller sampled dataset, which represents a smaller kernel matrix. Our final runtime is independent of n, the size of the dataset, whereas the prior state of the art result of [24] have a $\omega(n)$ runtime.

Kernel matrix low-rank approximation. In this setting, our goal here is to output a matrix B such that

$$\|K-B\|_F^2 \le \|K-K_t\|_F^2 + \varepsilon\|K\|_F^2$$

where K_t is the best rank-t approximation to the kernel matrix K. The efficient algorithm of [87] is able to achieve this guarantee if one can sample the ith row r_i of K with probability $p_i \ge \Omega(1)\cdot\|r_i\|_2^2/\|K\|_F^2$. We can perform such an action using our primitive, which is capable of sampling the rows of K with probability proportional to the squared row norms for the Laplacian, exponential, and Gaussian kernels. Thus for these kernels, we can immediately obtain efficient algorithms for computing a low-rank approximation.

7.2 Further Related Works

Remark 7.2.1. Spectral sparsification for kernel graphs has also been studied in prior works, notably in [10] and [166]. We first compare to [10], who obtain a spectral sparsification using an entirely different approach. They obtain an almost linear time sparsifier

$(n^{1+o(1)})$ when the kernel is *multiplicativily Lipschitz* (see Section 1.1.2 in [10] for definition) and show hardness for constructing such a sparsifier when it is not. Focusing on the Gaussian kernel, under Parameterization 1.6.1, [10] obtain an algorithm that runs in time $O\left(nd + n2^{\sqrt{\log(1/\tau)\log n}\log(\log n)}/\varepsilon^2\right)$, whereas our algorithm runs in $O\left(nd\log^2(n)/(\varepsilon^2\tau^{2.0173+o(1)})\right)$ time. We also note that the dimension d can be upper bounded by $O(\log n/\varepsilon^2)$ by applying Johnson-Lindenstrauss to the initial dataset. Therefore, [10] obtain a better dependence on $1/\tau$, whereas we obtain a better dependence on n. A similar comparison can be established for other kernels as well. In practice, τ is set to be a small fixed constant, whereas n can be arbitrarily large. Indeed in practice, a common setting of τ is 0.01 or 0.001, irrespective of the size of the dataset [23, 24, 122, 147, 177].

We now compare our guarantees to that of [166]. The author studies spectral sparsification resurrected to *smooth* kernels (for example kernels of the form $1/\|x-y\|_2^t$ which have a polynomial decay; see [166] for a formal definition). This family *does not* include Gaussian, Laplacian, or exponential kernels. For smooth kernels, [166] obtained a sparsifier with a nearly optimal $\tilde{O}(n/\varepsilon^2)$ number of edges in time $\tilde{O}(nd/\varepsilon^2)$. Our algorithm obtains a similar dependence in n, d, ε but includes an additional $1/\tau^3$ factor. However, it generalizes for any kernel supporting a KDE data structure, which includes smooth kernels [22] (see Table 2.1 for a summary of kernels where our results apply). Our techniques are also different: [166] does not use KDE data structures in a black-box manner to compute the sparsification as we do. Rather, they simulate importance sampling on the edges of the kernel graph directly. In addition to the nearly linear sparsifier, another interesting feature of [166] is that it enriches the connections between spectral sparsification of kernel graphs and KDE data structures. Indeed, the data structures used in [166] are inspired by and were used in the prior work of [22] to create KDE query data structures themselves. Furthermore, the paper demonstrates how to instantiate KDE data structures for smooth kernels using the kernel graph sparsifier itself. We refer to [166] for details.

Remark 7.2.2. Our algorithm returns a *sparse* vector v supported on roughly $O(1/(\varepsilon^2\tau^2))$ coordinates. The best prior result is that of [24] which presented an algorithm with total runtime $O\left(\frac{dn^{1+p}\log(n/\varepsilon)^{2+p}}{\varepsilon^{7+4p}}\right)$.

In comparison, our bound has *no dependence* on n and is thus a *truly sublinear* runtime. Note that the bound of [24] does not depend on τ. We do not state the number of KDE queries used explicitly in Table 1.1 since our algorithm uses KDE queries on a subsampled dataset and in addition, only uses them by calling the algorithm of [24] as a subroutine (on the subsampled dataset). The algorithm of [24] uses $\tilde{O}(1/\varepsilon)$ KDE queries but with various different initialization of τ so it is not meaningful to state "one" bound for the number of KDE queries used and thus the final runtime is a more meaningful quantity to state. Lastly, the authors in [24] present a lower bound of $\Omega(nd)$ for estimating the top eigenvalue λ_1, which ostensibly seems at odds with our stated bound which has no dependence on n. However, the lower bound presented in [24] essentially sets $\tau = 1/\operatorname{poly}(n)$ for a large polynomial factor depending on n (we estimate this factor to be $\Omega(n^2)$). Since we parameterize our dependence via τ, which in practice is often set to a fixed constant, we can bypass the lower bound.

Remark 7.2.3. We now compare our low-rank approximation result with a recent work of [28, 155]. They showed the following theorem:

Theorem 7.2.1 (Theorem 4.2, [28])**.** *Given a $n \times n$ PSD matrix A, target rank $r \in [n]$ and accuracy parameter $\varepsilon \in (0, 1)$, there exists an algorithm that queries $\widetilde{O}\left(nr/\varepsilon\right)$ entries in A and with probability at least $9/10$, outputs a rank-r matrix B such that*

$$\|A - B\|_F^2 \leq (1 + \varepsilon)\|A - A_r\|_F^2,$$

where A_r is the best rank-r approximation to A. Further, the running time is $\widetilde{O}\left(n\left(r/\varepsilon\right)^{\omega-1}\right)$, where ω is the matrix multiplication constant.

We note that their result applies to kernel matrices as well via the following fact.

Fact 7.2.2 (Kernel Matrices are PSD, [173])**.** *Let k be a reproducing kernel and X be n data points in $\mathbb{R}^d$. Let K be the associated $n \times n$ kernel matrix such that $K_{i,j} = k(x_i, x_j)$. Then, $K \succ 0$.*

Here, the family of reproducing kernels is quite broad and includes polynomial kernels, Gaussian, and Laplacian kernel, among others. Therefore, their theorem immediately implies a relative error low-rank approximation algorithm for kernel matrices. Our result and the theorem of [28] have comparable runtimes. While [28] obtain relative-error guarantees, we only obtain additive-error guarantees.

However, reading each entry of the kernel matrix require $O(d)$ time and thus [28] obtain an running time of $\widetilde{O}\left(nd\left(r/\varepsilon\right)^{\omega-1}\right)$, whereas our running time is dominated by $O(nrd/\varepsilon)$. We note that similar ideas as our algorithm for additive error LRA were previously used to design subquadratic algorithms running in time $o(n^2)$ for low-rank approximation of distance matrices [26, 112].

7.2.1 Preliminaries

First, we discuss the cost of constructing KDE data structure and performing the queries described in Definition 1.6.1. Table 2.1 summarizes previous work on kernel density estimation though for the sake of uniformity, we list only "high-dimensional" data structures, whose running times are polynomial in the dimension d. Those data structures have construction times of the form $O(dn/(\tau^p\varepsilon^2))$ and answer KDE queries in time $O(d/(\tau^p\varepsilon^2))$, under the condition that for all queries y we have $\frac{1}{n}\sum_{x\in X} k(x, y) \geq \tau$ (which clearly holds under our Parameterization 1.6.1). The algorithms are randomized, and report correct answers with a constant probability. The values of p lie in the interval $[0, 1)$, and depend on the kernel. For comparison, note that a simple random sampling approach, which selects a random subset $R \subset X$ of size $O(1/(\tau\varepsilon^2))$ and reports $\frac{n}{|R|}\sum_{x\in R} k(x, y)$, achieves the exponent of $p = 1$ for any kernel whose values lie in $[0, 1]$.

We view our algorithms as parameterized in terms of τ, the smallest edge length. We argue this is a natural parameterization. When picking a kernel function k, we also have to pick a *scale* term σ (for example, the exponential kernel is of the form $k(x, y) = \exp(-\|x - y\|_2/\sigma)$). In practice, a common choice of σ follows the so called 'median rule' where σ is set to be the median distance among all pairs of points in X. Thus, according to the median rule, the 'typical' kernel values in the graph K are $\Omega(1)$. While this is only true for 'typical,' and not all, edge weights in K, we believe the KDE query abstraction of Definition 1.6.1 still provides nontrivial and useful algorithms for working with kernel graphs. Typically in practice, the setting of τ is a small constant, independent of the size of the dataset [122].

We note that, in addition to the aforementioned algorithms with theoretical guarantees, there are other practical algorithms based on random sampling, space partition trees [89, 90, 133, 134, 147, 152, 168], coresets [159, 162, 208], or combinations of these methods [122], which support queries needed in Definition 1.6.1; see [122] for an in-depth discussion on applied works.

While these algorithms do not necessarily have as strong theoretical guarantees as the ones discussed above and in Table 2.1, we can nonetheless use them via black box access in our algorithms and utilize their practical benefits.

7.3 Algorithmic Building Blocks

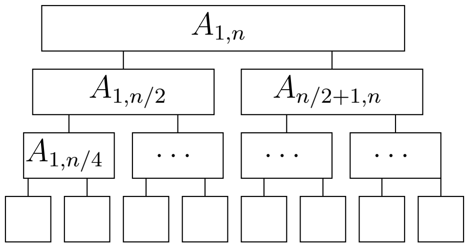


Figure 7.1: Multi-level Kernel Density Estimation Data Structure.

7.3.1 Multi-level KDE

We first describe the "multi-level" KDE data structure, which is required in our algorithms. The data structure recursively constructs a KDE data structure on the entire dataset X, and then recursively partitions X into two halves, building a KDE data structure on each half. See Algorithm 22 for more details.

Algorithm 22 Multi-level KDE Construction

```
1: Input: Dataset $X \subset \mathbb{R}^d$, precision $\varepsilon > 0$.
2: procedure MULTI-LEVEL-KDE-CONSTRUCTION
3:     Let $T = X$.
4:     while $|T| > 1$ do
5:         Construct $\text{KDE}_X$ queries (see Definition 1.6.1).
6:         Recursively apply to $T[1 : \lfloor m/2 \rfloor]$ and $T[\lfloor m/2 \rfloor + 1 : m]$
7:     end while
8:     Output: All the data structures associated with the KDE query constructions
9: end procedure
```

Lemma 7.3.1. *Given a dataset $X \subset \mathbb{R}^d$, suppose the initialization of the KDE data structure defined in Definition 1.6.1 uses runtime $f(n, \varepsilon)$ for some function linear in n. Then the total construction time of Algorithm 22 is $f(n \log n, \varepsilon)$.*

Proof. The proof follows from the fact that at each recursive level, we do $O(f(n, \varepsilon))$ total work since f is linear in n and there are $O(\log n)$ levels. □

7.3.2 Weighted Vertex Sampling

We now discuss our fundamental primitives. The first one computes approximate weighted degrees for all vertices. Algorithm 25 then performs vertex sampling by their (weighted) degree.

Algorithm 23 Computing Approximate (Weighted) Degrees

```
1: Input: Dataset $X \subset \mathbb{R}^d$, precision $\varepsilon > 0$.
2: procedure VERTEX-SAMPLING
3:     for $i \in [1, n]$ do
4:         $p_i \leftarrow \text{KDE}_X(x_i) - (1 - \varepsilon)\, k(x_i, x_i)$
5:     end for
6:     Output: Reals $\{p_i\}_{i=1}^n$ such that $(1-\varepsilon)\deg(x_i) \le p_i \le (1+\varepsilon)\deg(x_i)$ for all $1 \le i \le n$
7: end procedure
```

Definition 7.3.1 (Weighted Vertex Sampling)**.** *The weighted degree of a vertex x_i with $i \in [n]$ is $w_i = \sum_{j \neq i} k(x_i, x_j)$. The goal of weighted vertex sampling is to output a vertex v such that $\Pr[v = x_i] = \frac{(1 \pm \varepsilon) w_i}{\sum_{j \in [n]} w_j}$ for all $i \in [n]$.*

This is a straightforward application of using n KDE queries to get the (weighted) vertex degree of all n vertices. Note that this only takes n queries and only has to be *done once*. Therefore, we can think of vertex sampling as a preprocessing step that uses $O(n)$ queries upfront and then allows for arbitrary access at any point in the future with no query cost.

Once we acquire $\{p_i\}_{i=1}^n$, we can perform a fast sampling procedure through the following algorithm, which we state in slightly more general terms.

Algorithm 24 Sample from Positive Array

```
1: Input: Array $A = [a_1, \cdots, a_n]$ with $a_i > 0$ for all $i$. Access to queries $A_{i,j} = \sum_{i \in [t,j]} a_t$ for $1 \le i \le j \le n$.
2: procedure Array-Sampling
3:     Let $T = A$.
4:     while $|T| > 1$ do
5:         Let $m = \text{len}(T)$.
6:         Let $a \leftarrow \sum(T[1 : \lfloor m/2 \rfloor])$ //Can be simulated using an $A_{i,j}$ query
7:         Let $b \leftarrow \sum(T[\lfloor m/2 \rfloor + 1 : m])$.
8:         If $\text{Unif}[0,1] \le a/(a+b)$, $T \leftarrow T[1 : \lfloor m/2 \rfloor]$
9:         Else $T \leftarrow T[\lfloor m/2 \rfloor + 1 : m]$.
10:    end while
11:    Output: The single remaining element in $T$
12: end procedure
```

Combining Algorithms 23 and 24, we can sample from the degree distribution of the graph K.

Algorithm 25 Degree Sampling

```
1: Input: Dataset $X \subset \mathbb{R}^d$, precision $\varepsilon > 0$.
2: procedure Degree-Sampling
3:     Use Algorithm 23 to compute reals $\{p_i\}_{i=1}^n$ such that $(1-\varepsilon)\deg(x_i) \le p_i \le (1+\varepsilon)\deg(x_i)$ for all $1 \le i \le n$ (only needs to be done once).
4:     $i \leftarrow$ index in $[n]$, which is the output of running Algorithm 24 on the array $\{p_i\}_{i=1}^n$.
5:     Output: $x_i$ with probability $p_i / \sum_{j \in [n]} p_j$.
6: end procedure
```

We now analyze the correctness and the runtimes of the algorithms proposed in Section 7.3. First, we give guarantees on Algorithm 23.

Theorem 7.3.2. *Algorithm 23 returns $\{p_i\}_{i=1}^n$ such that $(1-\varepsilon)\deg(x_i) \le p_i \le (1+\varepsilon)\deg(x_i)$ for all $1 \le i \le n$.*

Proof. The proof follows by the Definition of a KDE query, Definition 1.6.1. □

We now analyze Algorithm 24, which samples from an array based on a tree data structure given access to consecutive sum queries. The analysis of this process will also greatly facilitate the analysis of other algorithms from Section 7.3.

Lemma 7.3.3. *Algorithm 24 samples an index $i \in [n]$ proportional to a_i in $O(\log n)$ time with $O(\log n)$ queries.*

Proof. Consider the sampling diagram given in Figure 7.1. Algorithm 24 does the following: it first queries the root node $A_{1,n}$ and then its two children $A_{1,m}, A_{m+1,n}$ where $m = \lfloor n/2 \rfloor$. Note that $A_{1,n} = A_{1,m} + A_{m+1,n}$. It then picks the tree rooted at $A_{1,m}$ with probability $\frac{\sum_{i\in[m]} a_i}{\sum_{i\in[n]} a_i}$ and otherwise, picks the tree rooted at $A_{m+1,n}$. The procedure recursively continues by querying the root node, its two children, and picking one of its children to be the new root node with probability proportional to the child's weight given by an appropriate query access. This is done until we reach a leaf node that corresponds to an index $i \in [n]$.

We now prove correctness. Note that each node of the tree in Figure 7.1 corresponds to a subset $S \subseteq [n]$. We prove inductively that the probability of landing on the vertex is equal to $\sum_{i\in S} a_i$. This is true for the root node of the tree since the algorithm begins at the root note. Now consider transitioning from some node S to one of its children S_1, S_2. We know that we are at node S with probability $\sum_{i\in S} a_i / \sum_j a_j$. Furthermore, we transition to S_1 with probability $\sum_{i\in S_1} a_i / \sum_{j\in S} a_j$. Therefore, the probability of being at S_1 is equal to

$$\frac{\sum_{i\in S_1} a_i}{\sum_{j\in S} a_j} \cdot \frac{\sum_{i\in S} a_i}{\sum_j a_j} = \frac{\sum_{i\in S_1} a_i}{\sum_j a_j}.$$

Since there is only one path from the root node to any vertex of a tree, this completes the induction.

The runtime and the number of queries taken follows from the fact that the sampling procedure descends on a tree with $O(\log n)$ height. □

Combining Algorithms 23 and 24 allows us to sample from the degree distribution of the graph K up to low error in total variation (TV) distance.

Theorem 7.3.4. *Algorithm 25 samples from the degree distribution of K up to TV error $O(\varepsilon)$ using a fixed overhead of n KDE queries and runtime $O(\log n)$.*

Proof. Since p_i is with a $1 \pm \varepsilon$ factor of $\deg(x_i)$ for all i, then $\{p_i\}_{i=1}^n$ is $O(\varepsilon)$ close in total variation distance from the true degree distribution. Moreover, Algorithm 24 perfectly samples from the array $\{p_i\}_{i=1}^n$, which proves the first part of the theorem.

For the second part, note that acquiring $\{p_i\}_{i=1}^n$ requires n KDE queries. We can then construct the data structure for Algorithm 24 by computing all the partial prefix sums in $O(n)$ time. Now the query access required by Algorithm 24 can be computed in $O(1)$ time through an appropriate subtraction of two prefix sums. Note that the previous steps need to be only done *once* and can be utilized for all future runs of Algorithm 24. It follows from Lemma 7.3.3 that Algorithm 25 takes $O(\log n)$ time. □

7.3.3 Weighted Edge Sampling and Weighted Neighbor Edge Sampling

We describe how to perform weighted neighbor edge sampling.

Definition 7.3.2 (Weighted Neighbor Edge Sampling)**.** *Given a vertex x_i, the goal of weighted neighbor edge sampling is to output a vertex v such that* $\Pr[v = x_k] = \frac{(1\pm\varepsilon)k(x_i,x_k)}{\sum_{j\in n, j\neq i} k(x_i,x_j)}$ *for all $i \in [n]$.*

Algorithm 26 Sample Random Neighbor

```
1: **Input:** Dataset $X \subset \mathbb{R}^d$, precision $\varepsilon > 0$, input vertex $x_i \in X$.
2: **procedure** NEIGHBOR-SAMPLING
3:     Let $\varepsilon' = \varepsilon/\log n$ and $T \leftarrow X \setminus \{x_i\}$.
4:     **while** $|T| > 1$ **do**
5:         Let $m \leftarrow |T|$.
6:         Compute $a \leftarrow \mathrm{KDE}_{T[1:m/2],\varepsilon'}(x_i)$ and $b \leftarrow \mathrm{KDE}_{T[m/2+1:m],\varepsilon'}(x_i)$.
7:         If $x_i \in T[1 : m/2]$, set $a \leftarrow a - (1-\varepsilon')k(x_i,x_i)$.
8:         If $x_i \in T[m/2+1 : m]$, set $b \leftarrow b - (1-\varepsilon')k(x_i,x_i)$.
9:         If Unif$[0,1] \le a/(a+b)$, let $T \leftarrow T[1 : m/2]$. Else, let $T \leftarrow T[m/2+1 : m]$.
10:     **end while**
11:     **Output:** Return the last element $x \in T$ such that $x \in X \setminus \{x_i\}$ and the probability
    of selecting $x$ is proportional to $k(x_i, x)$.
12: **end procedure**
```

We now prove the correctness of Algorithm 26 based on the ideas in Lemma 7.3.3. Note that Algorithm 26 takes in input a precision level ε, which can be adjusted and impacts the accuracy of KDE queries. We will discuss the cost of initializing KDE queries with various precisions in Section 7.2.1.

Theorem 7.3.5. *Let $x_i \in X$ be an input vertex. Consider the distribution $\mathcal{D}$ over $X \setminus \{x_i\}$, the neighbors of x_i in the graph K, induced by the edge weights in K. Algorithm 26 samples a neighbor from a distribution that is within TV distance $O(\varepsilon)$ from $\mathcal{D}$ using $O(\log n)$ KDE queries and $O(\log n)$ time.*

Proof. The proof idea is similar to that of Lemma 7.3.3. Given a vertex x_i, its adjacent edges have associated weights and our goal is to sample an edge proportion to these weights. However, unlike the degree case, performing edge sampling is not a straightforward KDE query as an edge only cares about the kernel value between two points, rather than the sum of kernel values that a KDE query provides. Nevertheless, we can utilize the tree procedure outline in the proof of Lemma 7.3.3 in conjunction with KDE queries with over various subsets of X.

Imagine the same tree as in Figure 7.1 where each subset corresponds to a subset of neighbors of x_i (note that x_i cannot be its own neighbor and hence we subtract $k(x_i, x_i)$ in line 7 or line 10). Algorithm 26 descends down the tree using the same probabilistic procedure as in the proof of Lemma 7.3.3: at every node, it picks one of the children to descend to with probability proportional to its weight. Here, the weight of a child node in the tree in Figure 7.1 is the sum of the weights of the edges connecting to the corresponding neighbors of x_i.

Now compare the telescoping product of probabilities that lands us in some leaf node a_j to the ideal telescoping product if we knew the exact array of edge weights as in the proof of Lemma 27. Suppose the tree has height ℓ. At each node in our actual path descending down the tree, we take the next step according to the ideal descent (according to the ideal telescoping product), with the same probability, except for possibly an overestimate or underestimate by a factor of $1+\varepsilon'$ or $1-\varepsilon'$ factor respectively.

Therefore, we land in the correct leaf node with the same probability as in the ideal telescoping product, except our probability can be off by a multiplicative $(1 \pm \varepsilon')^\ell$ factor. However, since $\varepsilon' = \varepsilon / \log n$ and $\ell \le \log n$, this factor is within $1 \pm \varepsilon$. Thus, we sample from the correct distribution over the leaves of the trees in Figure 7.1 up to TV distance $O(\varepsilon)$. □

Algorithm 27 Sample Random Edge by Weight

```
1: **Input:** Dataset $X \subset \mathbb{R}^d$, precision $\varepsilon > 0$.
2: **procedure** EDGE-SAMPLING
3:     Compute $x_i \leftarrow$ random vertex by using Algorithm 25.
4:     Compute $x_j \leftarrow$ random Neighbor of $x_i$ using Algorithm 26.
5:     **Output:** Edge $(x_i, x_j)$ such that $(x_i, x_j)$ is sampled with probability at least $(1 - \varepsilon)k(x_i, x_j)$.
6: **end procedure**
```

Theorem 7.3.6 (Weighted Edge Sampling)**.** *Algorithm 27 returns a random edge of K with probability proportional to at least $(1-\varepsilon)$ its weight using 1 call to Algorithm 26.*

Proof. Consider an edge (u, v). Vertex u is sampled with probability at least $(1-2\varepsilon)\frac{\deg(u)}{\sum_{x \in X} \deg(x)}$. Given this, v is then sampled with probability at least $(1-2\varepsilon)\frac{k(u,v)}{\sum_{x \in X \setminus u} k(u,x)} = (1-2\varepsilon)\frac{k(u,v)}{\deg(u)}$. Using the same analysis for sampling v and then u, we have that any edge (u, v) is sampled with probability at least $1 - 2\varepsilon$ times $k(u,v)/\sum_{e \in K} w(e)$. □

7.3.4 Random Walk

Theorem 7.3.7. *Algorithm 28 outputs a vertex from a vertex within $O(T\varepsilon)$ total variation distance from the true random walk distribution. Each step of the walk requires 1 call to Algorithm 26 .*

Proof. The proof follows from the correctness of Algorithm 26 given in Theorem 7.3.5. □

Algorithm 28 Perform Random Walk

```
1: Input: Dataset $X \subset \mathbb{R}^d$, precision $\varepsilon > 0$.Dataset $X \subset \mathbb{R}^d$, vertex $x_i \in X$, length of walk $T \geq 1$.
2: procedure RANDOM-WALK
3:     Start at vertex $v \leftarrow x_i$.
4:     for  $j = 1$ to $T$ do
5:         Sample a random neighbor of $v$ using Algorithm 26. Let $w$ be the resulting output.
6:         Set $v \leftarrow w$.
7:     end for
8:     Output: Data point $v$.
9: end procedure
```

7.4 Linear Algebra Applications

We now present applications of the algorithmic building blocks constructed in Section 7.3 to linear algebra.

7.4.1 Spectral Sparsification

Algorithm 29 Spectral Sparsification of the Kernel Graph

```
1: Input: Dataset $X \subset \mathbb{R}^d$, accuracy parameter $\varepsilon$.
2: procedure [SPECTRAL-SPARSIFICATION
3:     Let $t = O(n \log(n)/\varepsilon^2\tau^3)$ be the number of edges that are to be sampled
4:     Let $\hat{p}$ denote the distribution returned by Algorithm 23 for a small enough constant $\varepsilon$.
5:     Initialize $G' = \emptyset$.
6:     for $i = 1, \dots, t$ do
7:         Sample a vertex $u$ from the distribution $\hat{p}$.
8:         Sample a neighbor $v$ of $u$ using Algorithm 26 with constant $\varepsilon$.
9:         Compute $\hat{q}_{uv}$, the probability that Algorithm 26 samples $v$ given $u$ as input.
10:        Similarly define and compute $\hat{q}_{vu}$. Let $w_{uv} = 1/(t(\hat{p}_u\hat{q}_{uv} + \hat{p}_v\hat{q}_{vu}))$.
11:        Add the weighted edge $(\{u, v\}, w_{uv})$ to the graph $G'$.
12:    end for
13: end procedure
```

Given a set X, $|X| = n$, and a kernel $k : X \times X \to \mathbb{R}^+$, we describe how to construct a spectral sparsifier for the weighted complete graph on X where weight of the edge $\{x_i, x_j\}$ is given by $k(x_i, x_j)$.

Definition 7.4.1 (Graph Laplacian)**.** *Given a weighted graph $G = (V, E, w)$, the Laplacian of G, denoted by $L_G = D - A$, where A is the adjacency matrix of G with $A_{i,j} = w(\{i,j\})$ and D is a diagonal matrix such that for all $i \in [n]$, $D_{i,i} = \sum_{j \neq i} A_{i,j}$.*

Theorem 7.4.1 (Spectral Sparsification of Kernel Density Graphs)**.** *Given a dataset X of n points in $\mathbb{R}^d$, and a kernel $k : X \times X \to \mathbb{R}^+$, let $G = (X, \binom{X}{2}, w)$ be the weighted complete graph on X with the weights $w(\{x_i, x_j\}) = k(x_i, x_j)$. Further, for all $x_i, x_j \in X$, let $k(x_i, x_j) \geq \tau$, for some $\tau \in (0,1)$. Let L_G be the Laplacian matrix corresponding to the graph G. Then, for any $\varepsilon \in (0,1)$, Algorithm 29 outputs a graph G' with only $m = O(n \log n / (\varepsilon^2 \tau^3))$ edges, such that with probability at least $9/10$,*

$$(1-\varepsilon)L_G \preceq L_{G'} \preceq (1+\varepsilon)L_G.$$

The algorithm makes $\widetilde{O}(m/\tau^3)$ KDE queries and requires $\tilde{O}(md/\tau^3)$ post-processing time.

Let G_d be the weighted directed graph obtained by arbitrarily orienting the edges of the graph G and let H be an edge-vertex incidence matrix defined as follows : for each $e = (x_i, x_j)$ in graph G_d, let $H_{e,x_i} = \sqrt{k(x_i, x_j)}$ and $H_{e,x_j} = -\sqrt{k(x_i, x_j)}$. Note that $H^\top H = L_G$. Our idea to construct spectral sparsifier is to compute a sampling-and-reweighting matrix S, i.e., a matrix that has at most one nonzero entry in each row, that with probability $\geq 9/10$, satisfies

$$(1-\varepsilon)L_G = (1-\varepsilon)H^\top H \preceq H^\top S^\top S H \preceq (1+\varepsilon)H^\top H = (1+\varepsilon)L_G.$$

The edges sampled by S form the edges of the graph G'. We construct this matrix S by sampling rows of the matrix H from a distribution close to the distribution that samples a row of H with a probability proportional to its squared norm. We show that this gives a spectral sparsifier by showing that such a distribution approximates the "leverage score sampling" distribution.

Definition 7.4.2 (Leverage Scores)**.** *Let M be a $n \times d$ matrix and m_i denote the i-th row of M. Then, for all $i \in [n]$, τ_i, the i-th leverage of M is defined as follows:*

$$\tau_i = m_i (M^\top M)^+ m_i^\top,$$

where X^+ is the Moore-Penrose pseudoinverse for a matrix X.

We introduce the following intermediate lemmas. We begin by recalling that sampling edges proportional to leverage scores (effective resistances on a graph) suffices to obtain spectral sparsification [182, 200].

Lemma 7.4.2 (Leverage Score Sampling implies Sparsification)**.** *Given an $n \times d$ matrix M and $\varepsilon \in (0,1)$, for all $i \in [t]$, let τ_i be the i-th leverage score of M. Let $p = \{p_1, p_2, \ldots, p_n\}$ be a distribution over the rows of M such that $p_i = \tau_i / \sum_{j \in [n]} \tau_j$. Further, for some $\phi \in (0,1)$, let $\hat{p} = \{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n\}$ be a distribution such that $\hat{p}_i \geq \phi p_i$ and let $t = O\left(\frac{d \log(d)}{\varepsilon^2 \phi}\right)$. Let*

$S \in \mathbb{R}^{t \times n}$ be a random matrix where for all $j \in [t]$, the j-th row is independently chosen as $(1/\sqrt{t\hat{p}_i})e_i^\top$ with probability $\hat{p}_i$. Then, with probability at least $99/100$,

$$(1-\varepsilon)M^\top M \preceq M^\top S^\top S M \preceq (1+\varepsilon)M^\top M.$$

Next, we show that the matrix H is well-conditioned, in fact the condition number is independent of the dimension and only depends on the minimum kernel value between any two points in the dataset. This lets us use our edge sampling routines to compute an ε spectral sparsifier.

Lemma 7.4.3 (Bounding Condition Number)**.** *Let H be the edge-vertex incidence matrix as defined and also has the property that all nonzero entries in the matrix have an absolute value of at most 1 and at least $\sqrt{\tau}$. Let $\sigma_{\max}(H)$ be the maximum singular value of H and $\sigma_{\min}(H)$ be the minimum nonzero singular value of H. Then $\sigma_{\max}(H)/\sigma_{\min}(H) \le 4\sqrt{2}/\tau^{1.5}$.*

Proof. We use the following standard upper bound on the spectral norm of an arbitrary matrix A to upper bound the spectral norm of the matrix H:

$$\|A\|_2 \le \sqrt{\left(\max_i \sum_j |A_{i,j}|\right)\left(\max_j \sum_i |A_{i,j}|\right)}.$$

For the matrix H, as each column has at most n nonzero entries and each row has at most 2 non-zero entries and from the assumption that all the entries have magnitude at most 1, we obtain that $\|H\|_2 \le \sqrt{2n}$. To obtain lower bounds on $\sigma_{\min}(H)$, we appeal to a Cheeger-type inequality for weighted graphs from [85, 86]. First, we note that $\sigma_{\min}(H) = \sqrt{\sigma_{\min}(H^\top H)} = \sqrt{\sigma_{\min}(L_G)}$ where G is the kernel graph that we are considering with each edge having a weight of at least τ. Let $0 = \lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$ be the eigenvalues of the positive semi-definite matrix L_G. Now we have that

$$\sigma_{\min}(L_G) = \lambda_2(L_G) \ge \min_i (\delta_i/2)\varepsilon(G)^2$$

where $\delta_i = \sum_{j \ne i} k(x_i, x_j)$ i.e., the weighted degree of vertex x_i in graph G and

$$\varepsilon(G) = \min_{\phi \ne U \subset V, |U| \le n/2} \frac{|E(U, \bar{U})|}{|E(U)|}$$

where $|E(U)|$ denotes the sum of weighted degrees of vertices in U and $|E(U, \bar{U})|$ denotes the total weight of edges with one end point in U and the other outside U. Using the fact that G is a complete graph with each edge having a weight of at least τ and at most 1, we obtain $|E(U, \bar{U})| \ge \tau|U||\bar{U}|$ and $|E(U)| \le n|U|$, which implies that $\varepsilon(G) \ge \min_{\phi \ne U \subset V, |U| \le n/2} \tau|\bar{U}|/n \ge \tau/2$. We also similarly have that $\min_i \delta_i \ge (n-1)\tau$, which overall implies that $\lambda_2(L_G) \ge n\tau^3/16$ and that $\sigma_{\min}(H) \ge \sqrt{n}\tau^{1.5}/4$. Thus, we obtain that $\sigma_{\max}(H)/\sigma_{\min}(H) \le 4\sqrt{2}/\tau^{1.5}$. □

We are now ready to complete the proof of our main theorem:

Proof of Theorem 7.4.1. Let $q = \{q_1, q_2, \ldots, q_{\binom{n}{2}}\}$ be a distribution over the rows of H such that for all edges $e = \{i, j\}$, $q_e \geq c\frac{\|H_{e,*}\|_2^2}{\|H\|_F^2} = \frac{k(x_i,x_j)}{\sum_{e'=\{i',j'\}} k(x_{i'},x_{j'})}$, for a fixed universal constant c.

Next, we show that this distribution is $\Theta(1/\kappa^2)$ approximation to the leverage score distribution for H. Let $H = U\Sigma V^\top$ be the "thin" singular value decomposition of H and therefore all the diagonal entries of Σ are nonzero. By definition $\tau_i = \|U_{i*}\|_2^2$. We have

$$\|h_i\|_2^2 = \|U_{i*}\Sigma V^\top\|_2^2 = \|U_{i*}\Sigma\|_2^2$$

where the equality follows from the fact that $V^\top$ has orthonormal rows. Now, $\|U_{i*}\Sigma\|_2^2 \geq \|U_{i*}\|_2^2\sigma_{\min}^2$ and $\|U_{i*}\Sigma\|_2^2 \leq \|U_{i*}\|_2^2\sigma_{\max}^2$. Therefore, for all $i \in \binom{n}{2}$, defining $\kappa = \sigma_{\min}/\sigma_{\max}$, we have

$$\frac{\tau_i}{\sum_j \tau_j} = \frac{\|U_{i*}\|_2^2}{\sum_j \|U_{j*}\|_2^2} \geq \frac{\|h_i\|_2^2/\sigma_{\max}^2}{\sum_j \|h_j\|_2^2/\sigma_{\min}^2} = \frac{1}{\kappa^2}\frac{\|h_i\|_2^2}{\|H\|_F^2}.$$

Then, we invoke Lemma 7.4.2 with $\phi = \Omega(1/\kappa^2)$ and conclude that sampling $t = O\left(\frac{n\log n}{\varepsilon^2\kappa^2}\right)$ rows of H results in a sparse graph G' with corresponding Laplacian $L_{G'}$ such that with probability at least 99/100,

$$(1-\varepsilon/2)L_G \preceq L_{G'} \preceq (1+\varepsilon/2)L_G.$$

Further, by Lemma 7.4.3, we can conclude $\kappa^2 \leq 32/\tau^3$ and thus sampling $t = O\left(\frac{n\log n}{\varepsilon^2\tau^3}\right)$ edges suffices.

We do not use Algorithm 27 to sample random edges from the perfect distribution to implement spectral sparsification as we cannot compute the exact sampling probability of the edge that is sampled. So, we first use Algorithm 25 with constant ε (say 1/2) to sample a vertex u and Algorithm 26 with constant ε (say 1/2) to sample a neighbor v of u. Note that Algorithms 25 and Algorithms 26 can be modified to also return the probabilities $\hat{p}_u$ and $\hat{q}_{vu}$ with which the vertex i and the neighbor j of i are sampled. We can further query the algorithms to return $\hat{p}_v$ and $\hat{q}_{uv}$. Now, $q_{\{u,v\}} = \hat{p}_u\hat{q}_{vu} + \hat{p}_v\hat{q}_{uv}$ is the probability with which this sampling process samples the edge $\{u, v\}$ and we have that $\hat{p}_u\hat{q}_{vu} + \hat{p}_v\hat{q}_{uv} \geq c\frac{k(x_u,x_v)}{\sum_{i\neq j} k(x_i,x_j)}$ and we use this distribution q to implement spectral sparsification as described above. As already seen (Theorem 7.3.5), to compute vertex sampling distribution $\hat{p}$, we use n KDE queries and for each neighbor sampling step, we use $O(\log n)$ KDE queries. Thus, we overall use $O(n\log^2 n/(\varepsilon^2\tau^3))$ constant approximate KDE queries to obtain an ε spectral sparsifier. $\square$

We can further compute another graph G'' with only $O(n/\varepsilon^2)$ edges by computing an $\varepsilon/2$ spectral sparsifier for G' using the spectral sparsification algorithm of Lee an Sun [136] (see Theorem 1.1). This procedure doesn't require any KDE queries and solely operates on the weighted graph G'. The overall running time is $O\left(\frac{n^{1+1/c}\log(n)}{\epsilon^{6+1/c}\tau^3}\right)$, for a large fixed constant c.

Hardness for spectral sparsification. We observe that we can use the lower bound from Alman et. al. to establish hardness in terms of τ from Parameterization 1.6.1. The lower bound we obtain is as follows:

Theorem 7.4.4 (Lower Bound for Spectral Sparsification under Parameterization 1.6.1)**.** *Let k be the Gaussian kernel and let X be dataset such that $\min_{x,y \in X} k(x,y) = \tau$, for some $1 > \tau > 0$. Then, any algorithm that with probability $9/10$ outputs an $O(1)$-approximate spectral sparsifier for the kernel graph associated with X, with $O(n^{2-\delta})$ edges, where $\delta < 0.01$ is a fixed universal constant, requires $\Omega\left(n \cdot 2^{\log(1/\tau)^{0.32}}\right)$ time, assuming the strong exponential time hypothesis.*

First, we begin with the definition of a multiplicatively-Lipschitz function:

Definition 7.4.3 (Multiplicatively-Lipschitz Kernels)**.** *A kernel k over a set X is (c,L)-multiplicatively Lipschitz if for any $\rho \in (1/c, c)$, and for any $x, y \in X$, $c^{-L}k(x,y) \leq k(\rho x, \rho y) \leq c^{L}k(x,y)$.*

We will require the following theorem showing hardness for sparsification when the kernel function is not multiplicatively-Lipschitz:

Theorem 7.4.5 (Theorem 8.3 [10])**.** *Let k be a function and X be a dataset such that k is not (c,L)-multiplicatively-Lipschitz on X for some $L > 1$ and $c = 1 + 2\log\left(10 \cdot 2^{L^{0.48}}\right)/L$. Then, there is no algorithm that returns a sparsifier of the kernel graph associated with X with $O(n^{2-\delta})$ edges, where $\delta < 0.01$ is a fixed universal constant, in less than $O\left(n \cdot 2^{L^{0.48}}\right)$ time, assuming the strong exponential time hypothesis.*

Proof of Theorem 7.4.4 . First, we show that for any $c > 1$, if $L < \log(1/\tau)(c-1)$, then the Gaussian kernel k is not (c,L)-multiplicatively Lipschitz. Let $z = \|x-y\|_2^2$ and let $f(z) = e^{-z}$. Observe, it suffices to show that there exists a z such that $f(cz) \leq c^{-L}f(z)$. Let z be such that $f(z) = e^{z} = \min_{x,y} k(x,y) = \tau$, i.e. $z = \log(1/\tau)$. Then,

$$f(c\log(1/\tau)) = e^{-c\log(1/\tau)},$$

and for $L < \log(1/\tau)(c-1)$

$$c^{-L}f(\log(1/\tau)) > e^{-c\log(1/\tau)}.$$

Then, applying Theorem 7.4.5 with $c = 1 + \frac{1}{\sqrt{L}}$, it suffices to conclude k is not (c,L)-multiplicatively Lipschitz when $L < \log^{2/3}(1/\tau)$, which concludes the proof. □

Solving Laplacian Systems Approximately

We describe how to approximately solve the Laplacian system $L_G x = b$ using the spectral sparsifier $L_{G'}$. First, we note the following theorem that states the running time and approximation guarantees of fast Laplacian solvers.

Theorem 7.4.6 ([123], [183]). *There is an algorithm that takes an input a graph Laplacian L of a graph with m weighted edges, a vector b, and an error parameter α and returns x such that with probability at least $99/100$,*

$$\|x - L^+ b\|_L \le \alpha \|L^+ b\|_L,$$

where $\|x\|_L = \sqrt{x^\top L x}$. The algorithm runs in time $\widetilde{O}(m \log(1/\alpha))$.

We have the following theorem that bounds the difference between solutions for the exact Laplacian system and the spectral sparsifier Laplacian.

Theorem 7.4.7. *Let L_G be the Laplacian of a connected graph G on n vertices and let $L_{G'}$ be the Laplacian of an ε-spectral sparsifier G' of graph G i.e.,*

$$(1-\varepsilon)L_G \preceq L_{G'} \preceq (1+\varepsilon)L_G,$$

for $\varepsilon < c$ for a small enough constant c. Then, for any vector b with $1^\top b = 0$, $\|L_G^+ b - L_{G'}^+ b\|_{L_G} \le 2\sqrt{\varepsilon}\|L_G^+ b\|_{L_G}$.

Proof. Note that for $\varepsilon < 1$, the graph G' also has to be connected and therefore the only eigen vectors corresponding to eigen value 0 of the matrices L_G and $L_{G'}$ are of the form $a \cdot 1$ for $a \neq 0$ and hence columns (and rows) of L_G span all vectors orthogonal to 1. Therefore $L_G L_G^+ = L_G^+ L_G = I - (1/n)11^\top$. Now,

$$\begin{aligned}
\|L_G^+ b - L_{G'}^+ b\|_{L_G}^2 &= b^\top (L_G^+ - L_{G'}^+) L_G (L_G^+ - L_{G'}^+) b \\
&= b^\top L_G^+ L_G L_G^+ b - b^\top L_{G'}^+ L_G L_G^+ b - b^\top L_{G'}^+ L_G L_G^+ b + b^\top L_{G'}^+ L_G L_{G'}^+ b \\
&\le b^\top L_G^+ b - b^\top L_{G'}^+ b - b^\top L_{G'}^+ b + \frac{1}{1-\varepsilon} b^\top L_{G'}^+ b
\end{aligned}$$

where in the last inequality, we used $L_G L_G^+ b = 1$ and that for any vector v, $v^\top L_G v \le \frac{1}{1-\varepsilon} v^\top L_{G'} v$. As the null spaces of both L_G and $L_{G'}$ are given by $\{a1 \mid a \in \mathbb{R}\}$, we also obtain that

$$(1-\varepsilon)L_G^+ \preceq L_{G'}^+ \preceq (1+\varepsilon)L_G^+$$

using which we further obtain that

$$\|L_G^+ b - L_{G'}^+ b\|_{L_G}^2 \le \left(\frac{2}{1-\varepsilon} - 2\right) b^\top L_{G'}^+ b \le \frac{2\varepsilon(1+\varepsilon)}{1-\varepsilon} b^\top L_G^+ b \le 4\varepsilon \|L_G^+ b\|_{L_G}^2.$$

Thus, $\|L_G^+ b - L_{G'}^+ b\|_{L_G} \le 2\sqrt{\varepsilon}\|L_G^+ b\|_{L_G}$. □

Therefore, if x is a vector such that $\|x - L_{G'} b\|_{L_{G'}} \le \alpha \|L_{G'}^+ b\|_{L_{G'}}$ obtained using the fast Laplacian solver, then

$$\begin{aligned}
\|x - L_G^+ b\|_{L_G}^2 &= \|x - L_{G'}^+ b + L_{G'}^+ b - L_G^+ b\|_{L_G}^2 \\
&\le 2(\|x - L_{G'}^+ b\|_{L_G}^2 + \|L_{G'}^+ b - L_G^+ b\|_{L_G}^2) \\
&\le \frac{2}{1-\varepsilon}\|x - L_{G'}^+ b\|_{L_G'}^2 + 4\varepsilon \|L_G^+ b\|_{L_G}^2.
\end{aligned}$$

Here we used the above theorem and the fact that $L_G \preceq (1/(1-\varepsilon))L_{G'}$. Now, $\|x - L_{G'}^+ b\|_{L_{G'}}^2 \leq \alpha^2 \|L_{G'}^+ b\|_{L_{G'}}^2$ and $\|L_{G'}^+ b\|_{L_{G'}}^2 = b^\top L_{G'}^+ L_{G'} L_{G'}^+ b = b^\top L_{G'}^+ b \leq (1+\varepsilon) b^\top L_G^+ b \leq (1+\varepsilon)\|L_G^+ b\|_{L_G}^2$, which finally implies that

$$\|x - L_G^+ b\|_{L_G}^2 \leq \left(\frac{2(1+\varepsilon)^2}{1-\varepsilon} \alpha^2 + 4\varepsilon \right) \|L_G^+ b\|_{L_G}^2.$$

Thus, using a ε spectral sparsifier G' with m edges, we can in time $\widetilde{O}(m \log(1/\varepsilon))$ can obtain a vector x such that $\|x - L_G^+ b\|_{L_G} \leq C\sqrt{\varepsilon}\|L_G^+ b\|_{L_G}$ for a large enough constant C.

7.4.2 Low-rank Approximation of the Kernel Matrix

We derive algorithms for low-rank approximations of the kernel matrix via KDE queries. We present a algorithm for additive error approximation and compare to prior work for relative error approximation.

We first recall the following two theorems. Let $A_{i,*}$ denote the ith row of a matrix A.

Theorem 7.4.8 ([87]). *Let $A \in \mathbb{R}^{n \times m}$ be any matrix. Let S be a sample of $O(k/\varepsilon)$ rows according to a probability distribution $(p_1, \ldots, p_n)$ that satisfies $p_i \geq \Omega(1) \cdot \|A_{i,*}\|_2^2 / \|A\|_F^2$ for every $1 \leq i \leq n$. Then, in time $O(mk/\varepsilon \cdot \text{poly}(k, 1/\varepsilon))$, we can compute from S a matrix $U \in \mathbb{R}^{k \times m}$, that with probability at least 0.99 satisfies*

$$\|A - AU^T U\|_F^2 \leq \|A - A_k\|_F^2 + \varepsilon \|A\|_F^2.$$

Theorem 7.4.9 ([63], also see [112]). *There is a randomized algorithm that given matrices $A \in \mathbb{R}^{n \times m}$ and $U \in \mathbb{R}^{k \times m}$, reads only $O(k/\varepsilon)$ columns of A, runs in time $O(mk) + \text{poly}(k, 1/\varepsilon)$, and returns $V \in \mathbb{R}^{n \times k}$ that with probability 0.99 satisfies*

$$\|A - VU\|_F^2 \leq (1+\varepsilon) \min_{X \in \mathbb{R}^{n \times k}} \|A - X\|_F^2.$$

Therefore to compute the low rank approximation, we just need sample from the distribution on rows required by Theorem 7.4.8. We reduce this question to evaluating KDE queries as follows: If K is the kernel matrix, each row of K is the weight of the edges of the corresponding vertex. Therefore, each p_i in the distribution $(p_1, \ldots, p_n)$ is the sum of edge weights *squared* for vertex x_i. From vertex queries (Algorithm 25), we know that we can get the degree of each vertex, which is the sum of edge weights. We can extend Algorithm 25 to sample from the sum of *squared* edge weights of each vertex as follows. Consider a kernel k such that there exists an absolute constant c that satisfies $k(x,y)^2 = k(cx, cy)$ for all x, y. Such a c exists for the most popular kernels such as the Laplacian, exponential, and Gaussian kernels for which $c = 2, 2$, and 4 respectively. Thus give our dataset X, we simply construct KDE queries for the dataset $X' := cX$. Then by sampling the degrees of the vertices associated with the kernel graph K' of X', we can sample from the distribution required by Theorem 7.4.8 by invoking Algorithm 25 on the dataset X'. In particular, using

n KDE queries for X', we can get row norm squared values for all rows of our original kernel matrix K. We can then sample the rows according to Theorem 7.4.8 and fully construct the rows that are sampled. Altogether, this takes n KDE queries and $O(nk/\varepsilon)$ kernel function evaluations to construct a rank k approximation of K; see Algorithm 30.

Corollary 7.4.10. *Given a dataset X of size n, there exists an algorithm that outputs a rank k matrix B such that*

$$\|K - B\|_F^2 \le \|K - K_k\|_F^2 + \varepsilon\|K\|_F^2$$

with probability $99/100$, where K is a kernel matrix associated with X based on a Laplacian, exponential, or Gaussian kernel, and K_k is the optimal rank-k approximation of K. It uses n KDE queries and $O(nk/\varepsilon \cdot \mathrm{poly}(k, 1/\varepsilon) + nkd/\varepsilon)$ post-processing time.

We remark that for the application presented in this subsection, we can we can replace 1.6.1. Indeed, since we only estimate row sums, we only require that the value of a KDE query is at least τ, that is, the average value $\frac{1}{|X|}\sum_{x\in X} k(x,y) \ge \tau$ for a query y. Note that via Cauchy Schwartz, this automatically implies a lower bound for the average squared sum:

$$\frac{1}{|X|}\sum_{x\in X} k(x,y)^2 \ge \frac{1}{|X|^2}\left(\sum_{x\in X} k(x,y)\right)^2 \ge \tau^2.$$

Algorithm 30 Additive-error Low-rank Approximation

```
1: Input:Kernel matrix $K \in \mathbb{R}^{n\times n}$, data points $X \subset R^d$, accuracy parameter $\varepsilon$, rank parameter $k$.
2: procedure ADDITIVE-LRA
3:     Let $c$ be the constant such that $k(x,y)^2 = k(cx, cy)$ for all inputs $x, y$.
4:     for $i = 1$ to $i = n$ do
5:         Compute the value $p_i = \sum_{j=1}^n k(cx_i, cx_j)$ using KDE queries for the dataset $cX$.
6:     end for
7:     Sample and construct $O(k/\varepsilon)$ rows of $K$ according to probability proportional to $\{p_i\}_{i=1}^n$.
8:     Compute $U$ from the sample, using Theorem 7.4.8.
9:     Compute $V$ from the sample, using Theorem 7.4.9.
10:    Output: Factors $U, V$ such that $\|K - UV\|_F^2 \le \|K - K_k\|_F^2 + \varepsilon\|K\|_F^2$
11: end procedure
```

7.4.3 First Eigenvalue and Eigenvector Approximation

Our goal is to approximate the top eigenvalue of the kernel matrix and find a vector witnessing this approximation. Our overall algorithm can be split into two steps: first sample a random principal submatrix of the kernel matrix. Under the condition that each row of the $n \times n$

kernel matrix K satisfies that it's sum is at least $n\tau$, we can easily show that it must have a large first eigenvalue and thus prior works on sampling bounds automatically imply the first eigenvalue of the sampled matrix approximates that of K. The next step is to use a 'noisy' power method of [24] on the *sampled submatrix*. We note that this step employs a KDE data-structure initialized only on the sampled indices of K. The algorithm and details follow.

Algorithm 31 First Eigenvalue and Eigenvector Approximation

```
1: Input:Input dataset $X \subset \mathbb{R}^d$ of size $|X| = n$, precision $\varepsilon > 0$.
2: procedure FIRST-EIGENVALUE-EIGENVECTOR
3:     Let $t \leftarrow O(1/(\varepsilon^2\tau^2)$. Let $S \leftarrow$ random subset of $[n]$ of size $t$. Let $X_S$ be the samples restricted to the indices in $S$.
4:     Let $K_S \leftarrow$ principal submatrix of $K$ on indices in $S$ and let $\tilde{K} \leftarrow (n/s) \cdot K_S$. // Just for notation; we do not initialize $K$ or $K_S$
5:     Construct a KDE data structure for $X_S$. Run Algorithm 1 of [24] (Kernel Noisy Power Method) on $K_S$. Let $\hat{\lambda}_{\max}$ be the resulting eigenvalue and $\hat{v}_{\max}$ be the resulting eigenvector.
6:     Output: $\hat{\lambda}_{\max}$ and $\hat{v}_{\max}$.
7: end procedure
```

We remark that the eigenvector returned by Algorithm 31 will be a *sparse* vector supported only on the coordinates in S.

We first state the necessary auxiliary statements needed to prove the guarantees of Algorithm 31.

Lemma 7.4.11. *If each row of K satisfies that its sum is at least $n\tau$ for parameter $\tau \in (0,1)$, then the largest eigenvalue of K, denoted as λ_1, satisfies $\lambda_1 \geq n\tau$.*

Proof. This follows from looking at the quadratic form $\mathbf{1}^T K \mathbf{1}$ where $\mathbf{1}$ is the vector with all entries equal to 1:

$$\lambda_1 \geq \frac{\mathbf{1}^T K \mathbf{1}}{\mathbf{1}^T \mathbf{1}} \geq \frac{n^2\tau}{n} = n\tau. \qquad \square$$

We now state the guarantees of Algorithm 1 in [24].

Theorem 7.4.12 ([24])**.** *Suppose the kernel function for a $m \times m$ kernel matrix K has a KDE data structure with query time $d/(\varepsilon^2\tau^p)$ (see Table 2.1). Then Algorithm 1 of [24] returns λ such that $\lambda \geq (1-\varepsilon)\lambda_1(K)$ in time $O\left(\frac{dm^{1+p}\log(m/\varepsilon)^{2+p}}{\varepsilon^{7+4p}}\right)$*

Finally, we need the following result on eigenvalues of sampled PSD matrices, proven in [36].

Lemma 7.4.13 ([36])**.** *Let $A \in \mathbb{R}^{n\times n}$ be PSD with $\|A\|_\infty \leq 1$. Let $S \subset [n]$ be a random subset of size t and let $A_{S\times S}$ be the submatrix restricted to columns and rows in S and scaled by n/s. Then, for all $i \in [|S|]$, $\lambda_i(A_{S\times S}) = \lambda_i(A) \pm \frac{n}{\sqrt{t}}$.*

We are now ready to prove the guarantees of Algorithm 31.

Theorem 7.4.14. *Given a $n \times n$ kernel matrix K admitting a KDE data-structure with query time $d/(\varepsilon^2\tau^p)$, Algorithm 31 returns λ such that $\lambda \geq (1-\varepsilon)\lambda_1(K)$ in total time*

$$\min\left(O\left(\frac{d\log(d/\varepsilon)}{\varepsilon^{4.5}\tau^4}\right), O\left(\frac{d}{\varepsilon^{9+6p}\tau^{2+2p}}\log\left(\frac{1}{\varepsilon\tau}\right)^{2+p}\right)\right).$$

Remark 7.4.1. Two remarks are in order. First we recall that the runtime of [24] has a n^{1+p} factor while our bound *has no dependence* on n and is thus a truly sublinear runtime. Second, if we skip the Kernel Noisy Power method step and directly initialize and calculate the top eigenvalue of K_S (using the standard gap independent power method of [153]), we would get a runtime of $\tilde{O}(d/(\varepsilon^{4.5}\tau^4))$ which has a polynomially better ε dependence but a worse τ dependence than the guarantees of Algorithm 31.

Proof of Theorem 7.4.14. We first prove the approximation guarantee. By our setting of t and using Lemma 7.4.13, we see that the additive error in approximating the first eigenvalue of K by that of $\tilde{K}$ is at most

$$\frac{n}{\sqrt{t}} \leq \varepsilon\tau n \leq \varepsilon\lambda_1(K),$$

and thus $\lambda_1(\tilde{K}) \geq (1-\varepsilon)\lambda_1(K)$. Then by the guarantees of Theorem 7.4.12, it follows that we find a $1-\varepsilon$ multiplicative approximation to $\lambda_1(\tilde{K})$ and thus a $1-O(\varepsilon)$ multiplicative approximation to that of $\lambda_1(K)$.

We now prove the runtime bound. It easily follows from plugging in $m = O(1/(\varepsilon^2\tau^2))$ in Theorem 7.4.12. $\square$

7.5 Empirical Evaluation

We present empirical evaluations for our algorithms. We chose to evaluate algorithms for low-rank approximation and spectral sparsification as they are arguably two of the most well studied examples in our applications. For our experiments, we use the Laplacian kernel $k(x,y) = \exp(-\|x-y\|_1/\sigma)$. A fast KDE implementation of this kernel exists due to [23], which builds upon the techniques of [55]. Note that the focus of our work is to use KDE queries in a mostly black box fashion to solve important algorithmic problems for kernel matrices. This viewpoint has the important advantage that it is flexible to the choice of any particular KDE query instantiation. We chose to work with the implementation of [23] since it possesses theoretical guarantees, has an accessible implementation[1], and has been used in experiments in prior works such as [23, 24]. However, we envision other choices of KDE queries, which maybe have practical benefits but are theoretically incomparable would also work well due to our flexibility.

[1]from https://github.com/talwagner/efficient_kde

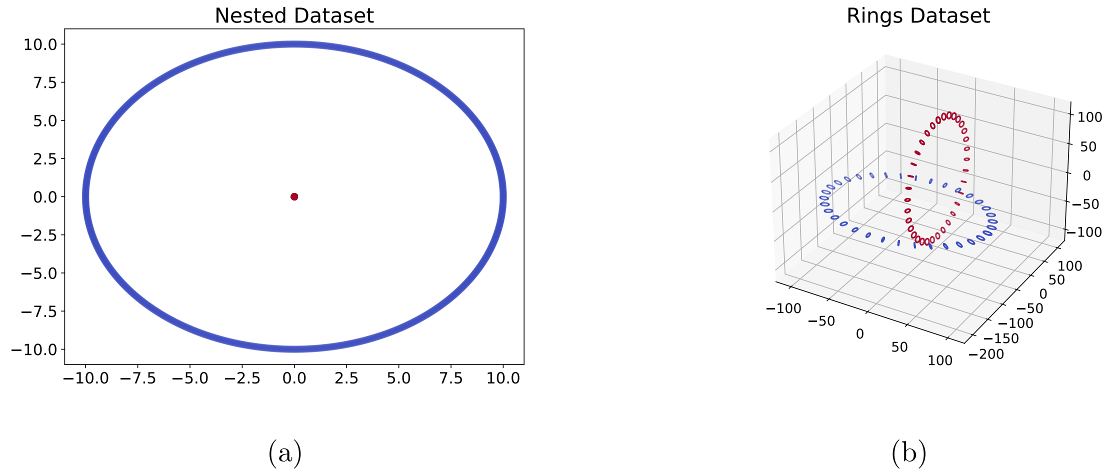


Figure 7.2: (a) Nested Dataset, (b) Rings Dataset

Datasets. We use two real and two synthetic datasets in our experiments. The datasets used in the low-rank approximation experiments are MNIST (points in $\mathbb{R}^{784}$) [132] and Glove word embeddings (points in $\mathbb{R}^{200}$)[158]. We use 10^4 points from each of their test datasets. These datasets have been used in prior experimental works on kernel density estimation [23, 177].

For spectral sparsification and clustering, we use construct two synthetic datasets, which are challenging for other clustering method such as k-means clustering[2]. The first dataset denoted as 'Nested' consists of $5,000$ points, equally split among the origin and a circle of radius 1. The two natural clusters are the points at the origin and the points on the circle. Since one cluster is contained in the convex hull of the other, a method like k-means clustering will not be able to separate the two clusters, but it is known that spectral clustering can. Our second dataset, labeled 'Rings', is an even more challenging clustering dataset. We consider two tori in three dimensions that pass through the interior hole of each other, i.e., they interlock. The 'small' radius of each tori is 5 while the 'large' radius is 100. Our dataset consists of 2500 points uniformly distributed on the two tori; see Figure 7.2b. Note that our focus is not to compare the efficacy of various clustering methods, which is done in other prior works (e.g., see footnote 2). Rather, we show that spectral clustering itself can be optimized in terms of runtime, space usage, and the number of kernel evaluations performed via our algorithms.

Evaluation metrics. For low-rank approximation, we use the additive error algorithm detailed in Corollary 7.4.10 of Section 7.4.2. It requires sampling the rows of the kernel matrix according to squared row norms, which can be done via KDE queries as outlined there. Once the (small) number of rows are sampled, we explicitly construct these rows using kernel evaluations. We compare the approximation error of this method computed via the

[2]For example, see https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html.

standard Frobenius norm error to a state of the art sketching algorithm for computing low-rank approximations, which is the input-sparsity time algorithm of Clarkson and Woodruff [67] **(IS)**. We also compare to an iterative SVD solver **(SVD)**. All linear algebra subroutines rely on Numpy and Scipy implementations and Numba complication when applicable.

Note that prior works such as [23, 24] have used use the number of kernel evaluations performed (how many entries of K are computed) as a measure of computational cost. While this is a software and architecture independent metric, which is unaffected by access to specialized libraries or hardware (e.g., SIMD, GPU), it is of interest to go beyond this measure. We use this measure as well as other important metrics as space usage and runtime as points of comparison. For spectral sparsification and clustering, we compare the accuracy of our method to the clustering solution when run on the full initialized kernel matrix.

Parameter settings. For low-rank approximation, we choose the bandwidth value σ according to the choice made in prior experiments in [23]. There, σ is chosen according to the popular median distance rule; see their experimental section for further information. For our clustering experiments, we pick the value of σ, which results in spectral clustering (running on the full kernel matrix) successfully clustering the input.

7.5.1 Results

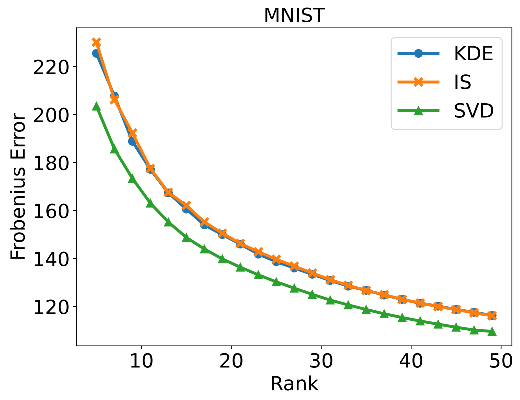
Low-rank approximation. Note that the algorithm in Corollary 7.4.10 has a $O(k)$ dependence on the number of rows sampled. Concretely we sample $25k$ rows for a rank k approximation which we fix it for all experiments. For the MNIST dataset, the rank versus approximation error is shown in Figure 7.3a. The performance of our algorithm labeled as **KDE** is given by the blue curve while the orange curve represents the **IS** algorithm. The green curve represents the SVD error, which is a lower bound on the error for any algorithm. Note that for SVD calculations, we do not calculate the full SVD since that is computationally prohibitive; instead, we use an iterative solver. We can see that the errors of all three methods are comparable to each other. In terms of runtime, the KDE based method took 24.7 seconds on average for the rank 50 approximation whereas **IS** took 71.5 seconds and iterative SVD took 74.72 seconds on average. This represents a **2.9x** decrease in the running time. The time measured includes the time to initialize the data structures and matrices used for the respective algorithms. In terms of the number of kernel evaluations, both **IS** and iterative SVD require the kernel matrix, which is 10^8 kernel evaluations. On the other hand for the rank 50 approximation, our method required only $1.1 \cdot 10^7$ kernel evaluations, which is a **9x** decrease in the number of evaluations. In terms of space, **IS** and iterative SVD require 10^8 floating point numbers stored due to initializing the full $10^4 \times 10^4$ matrix whereas our method only requires $10^4 \cdot 25 \cdot 50$ floating point numbers for the rank equal to 50 case and smaller for other. This is a **8x** decrease in the space required. Lastly, we verify that we are indeed sampling from the correct distribution required by Corollary 7.4.10. In Figure 7.3b, we plot the points (x_i, y_i) where x_i is the row norm squared for the ith row of the kernel matrix K and y_i is the row norm squared computed in our approximation algorithm (see Algorithm 30). As shown in Figure 7.3b, the data points fall very close to the $y = x$ line indicating that our algorithm is indeed sampling from approximately the correct ideal distribution.

The qualitatively similar results for the Glove dataset are given in Figures 7.3c and 7.3d. For the glove dataset, the average time taken by the three algorithms were $37.7, 37.7$, and 44.2 seconds respectively, indicating that **KDE** and **IS** were comparable in runtime whereas SVD took slightly longer. However, the number of kernel evaluations required by the latter two algorithms was significantly larger: for rank equal to 10, our algorithm only required $2.6 \cdot 10^6$ kernel evaluations while the other methods both required 10^8 due to initializing the matrix. Lastly, the space required by our algorithm was smaller by a factor of 40 since we only explicitly compute $25 \cdot 10$ rows for the rank $= 10$ case. For Glove, we only perform our experiments up to rank equal to 10 since the iterative SVD failed to converge for higher ranks. While computing the full SVD would avoid the convergence issue, it would take significantly longer time in general. For example for the MNIST dataset, computing the full SVD of the kernel matrix took 552.9 seconds, which is approximately an order of magnitude longer than any of the other methods.

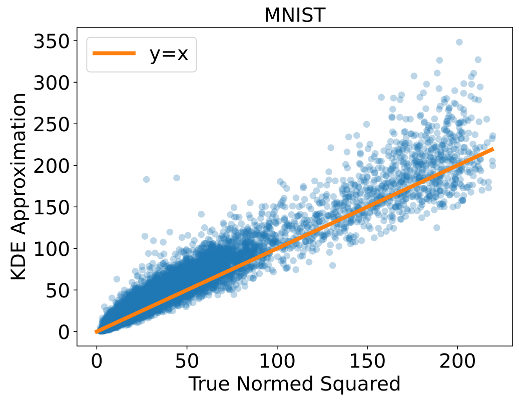
Spectral sparsification and clustering. Our algorithm consists of running the spectral sparsification algorithm of Theorem 7.4.1 (Algorithm 29) and computing the first two eigenvectors of the normalized Laplacian of the resulting sparse graph. We then run k-means clustering on the computed Laplacian embedding for $k = 2$.

As noted above, we use two datasets that pose challenges for traditional clustering methods such as k-means clustering. The Nested dataset is shown in Figure 7.2a. We sampled $3 \cdot 10^5$ many edges, which is 2.5% of total edges. Figure 7.4a shows the Laplacian embedding of the sampled graph based on the first two eigenvectors. The colors of the red and blue points correspond to their cluster in Figure 7.2a as identified by running k-means clustering on the Laplacian embedding. The orange crosses are the points that the spectral clustering method failed to correctly classify. These are only 23 points, which represent a 0.5% of total points. Furthermore, Figure 7.4a shows that the Laplacian embedding of the sampled graph is able to embed the two clusters into distinct and disjoint regions. Note that the total space savings of the sampled graph over storing the entire graph is **41x**. In terms of the time taken, the iterative SVD method used to calculate the Laplacian eigenvectors took 0.18 seconds on the sparse graph whereas the same method took 0.81 seconds on the entire graph. This is a **4.5x** factor reduction.

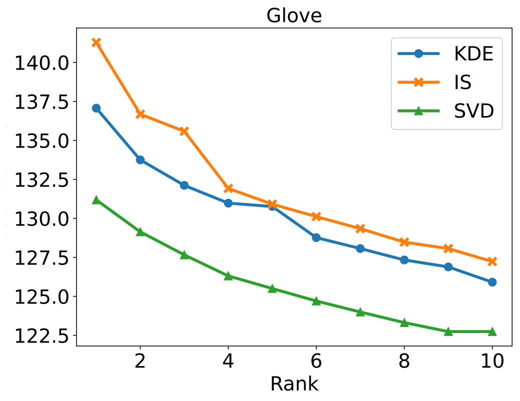
We recorded qualitatively similar results for the rings dataset. Figure 7.2b shows a plot of the dataset. We sampled 10^5 many edges for the approximation, which represents a 3.3% of total edges for form the sparse graph. The Laplacian embedding of the sparse graph is shown in Figure 7.4b. In this case, the embedding constructed from the sparse graph was able to separate the two rings into disjoint regions perfectly. The time taken for computing the Laplacian eigenvectors for the sparse graph was 0.08 seconds whereas it took 0.27 seconds for the full dense matrix.



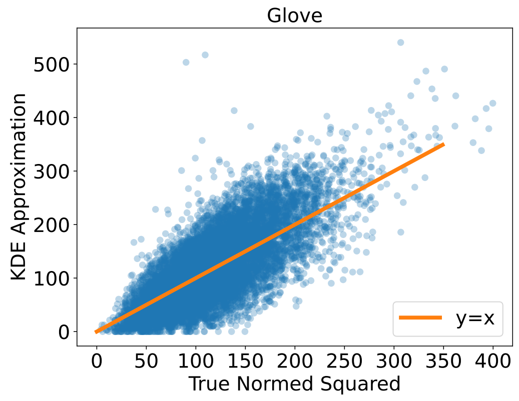
(a) Rank vs Error for Low-rank Approximation



(b) Real/Approximate Row Norm Squared

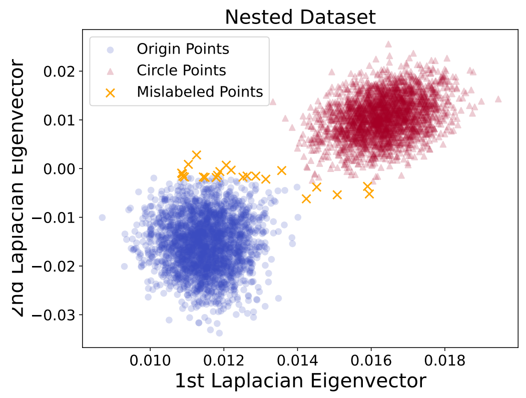


(c) Rank vs Error for Low-rank Approximation

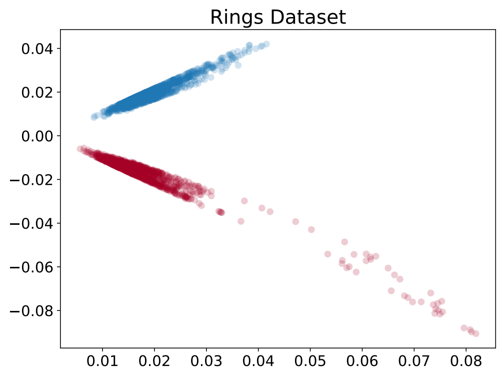


(d) Real/Approximate Row Norm Squared

Figure 7.3: Figures for low rank approximation experiments.



(a)



(b)

Figure 7.4: Spectral embedding of sparsified graph for (a) Nested dataset and (b) Rings dataset, respectively.

References

[1] Pdal: Chamfer. https://pdal.io/en/2.4.3/apps/chamfer.html, 2023. Accessed: 2023-05-12.

[2] Pytorch3d: Loss functions. https://pytorch3d.readthedocs.io/en/latest/modules/loss.html, 2023. Accessed: 2023-05-12.

[3] Tensorflow graphics: Chamfer distance. https://www.tensorflow.org/graphics/api_docs/python/tfg/nn/loss/chamfer_distance/evaluate, 2023. Accessed: 2023-05-12.

[4] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[5] Thomas D Ahle, Michael Kapralov, Jakob BT Knudsen, Rasmus Pagh, Ameya Velingker, David P Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 141–160. SIAM, 2020.

[6] Nir Ailon and Bernard Chazelle. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009. doi:10.1137/060673096. URL https://doi.org/10.1137/060673096.

[7] Francesco Aldà and Benjamin I. P. Rubinstein. The bernstein mechanism: Function release under differential privacy. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1705–1711. AAAI Press, 2017. doi:10.1609/aaai.v31i1.10884. URL https://doi.org/10.1609/aaai.v31i1.10884.

[8] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[9] Josh Alman, Timothy Chu, Aaron Schild, and Zhao Song. Algorithms and hardness for linear algebra on geometric graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 541–552. IEEE, 2020. doi:10.1109/FOCS46700.2020.00057. URL https://doi.org/10.1109/FOCS46700.2020.00057.

[10] Josh Alman, Timothy Chu, Aaron Schild, and Zhao Song. Algorithms and hardness

for linear algebra on geometric graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 541–552. IEEE, 2020.

[11] Noga Alon. Problems and results in extremal combinatorics—i. *Discrete Mathematics*, 273(1-3):31–53, 2003.

[12] Jason M. Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1964–1974, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/491442df5f88c6aa018e86dac21d3606-Abstract.html.

[13] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801, 2015.

[14] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3287–3318. World Scientific, 2018.

[15] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[16] Uri M Ascher and Chen Greif. *A first course on numerical methods*. SIAM, 2011.

[17] Kubilay Atasu and Thomas Mittelholzer. Linear-complexity data-parallel earth mover's distance approximations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 364–373. PMLR, 6 2019. URL https://proceedings.mlr.press/v97/atasu19a.html.

[18] Vassilis Athitsos and Stan Sclaroff. Estimating 3d hand pose from a cluttered image. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–432. IEEE, 2003.

[19] Haim Avron, Huy Nguyen, and David Woodruff. Subspace embeddings for the polynomial kernel. *Advances in neural information processing systems*, 27, 2014.

[20] Artem Babenko and Victor Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2055–2063, 2016.

[21] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. On the fine-grained complexity of empirical risk minimization: Kernel methods and neural networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 4308–4318, 2017.

[22] Arturs Backurs, Moses Charikar, Piotr Indyk, and Paris Siminelakis. Efficient density

evaluation for smooth kernels. *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 615–626, 2018.

[23] Arturs Backurs, Piotr Indyk, and Tal Wagner. Space and time efficient kernel density estimation in high dimensions. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, pages 15773–15782, 2019.

[24] Arturs Backurs, Piotr Indyk, Cameron Musco, and Tal Wagner. Faster kernel matrix algebra via density estimation. In *Proceedings of the 38th International Conference on Machine Learning*, pages 500–510, 2021.

[25] Arturs Backurs, Zinan Lin, Sepideh Mahabadi, Sandeep Silwal, and Jakub Tarnawski. Efficiently computing similarities to private datasets. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HMe5CJv9dQ.

[26] Ainesh Bakshi and David Woodruff. Sublinear time low-rank approximation of distance matrices. *Advances in Neural Information Processing Systems*, 31, 2018.

[27] Ainesh Bakshi, Nadiia Chepurko, and Rajesh Jayaram. Testing positive semi-definiteness via random submatrices. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1191–1202. IEEE, 2020.

[28] Ainesh Bakshi, Nadiia Chepurko, and David P Woodruff. Robust and sample optimal algorithms for PSD low rank approximation. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 506–516. IEEE, 2020.

[29] Ainesh Bakshi, Kenneth L. Clarkson, and David P. Woodruff. Low-rank approximation with $1/\epsilon^{1/3}$ matrix-vector products. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1130–1143. ACM, 2022. doi:10.1145/3519935.3519988. URL https://doi.org/10.1145/3519935.3519988.

[30] Ainesh Bakshi, Piotr Indyk, Praneeth Kacham, Sandeep Silwal, and Samson Zhou. Subquadratic algorithms for kernel matrices via kernel density estimation. In *The Eleventh International Conference on Learning Representations*, 2022.

[31] Ainesh Bakshi, Piotr Indyk, Rajesh Jayaram, Sandeep Silwal, and Erik Waingarten. Near-linear time algorithm for the chamfer distance. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d2fe3a5711a6d488da9e9a78b84ee24c-Abstract-Conference.html.

[32] Joshua Batson, Daniel A Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: theory and algorithms. *Communications of the ACM*, 56(8): 87–94, 2013.

[33] Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: beyond subspaces and the johnson-lindenstrauss lemma. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, pages 1039–1050, 2019.

[34] Amos Beimel, Hai Brenner, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. Bounds on the sample complexity for private learning and private data release. *Machine learning*, 94:401–437, 2014.

[35] Radu Berinde, Anna C Gilbert, Piotr Indyk, Howard Karloff, and Martin J Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 798–805. IEEE, 2008.

[36] Rajarshi Bhattacharjee, Gregory Dexter, Petros Drineas, Cameron Musco, and Archan Ray. Sublinear time eigenvalue approximation via random sampling. *Algorithmica*, pages 1–66, 2024.

[37] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. The johnson-lindenstrauss transform itself preserves differential privacy. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 410–419. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.67. URL https://doi.org/10.1109/FOCS.2012.67.

[38] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to noninteractive database privacy. *J. ACM*, 60(2):12:1–12:25, 2013. doi:10.1145/2450142.2450148. URL https://doi.org/10.1145/2450142.2450148.

[39] Christos Boutsidis, Anastasios Zouzias, and Petros Drineas. Random projections for k-means clustering. *Advances in neural information processing systems*, 23, 2010.

[40] Mark Braverman, Elad Hazan, Max Simchowitz, and Blake Woodworth. The gradient complexity of linear regression. In *Conference on Learning Theory*, pages 627–647. PMLR, 2020.

[41] Karl Bringmann, Nick Fischer, and Vasileios Nakos. Sparse nonnegative convolution is equivalent to dense nonnegative convolution. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1711–1724. ACM, 2021. doi:10.1145/3406325.3451090. URL https://doi.org/10.1145/3406325.3451090.

[42] Karl Bringmann, Nick Fischer, and Vasileios Nakos. Deterministic and las vegas algorithms for sparse nonnegative convolution. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3069–3090. SIAM, 2022. doi:10.1137/1.9781611977073.119. URL https://doi.org/10.1137/1.9781611977073.119.

[43] Bo Brinkman and Moses Charikar. On the impossibility of dimension reduction in l_1.

J. ACM, 52(5):766–788, 2005. doi:10.1145/1089023.1089026. URL https://doi.org/10.1145/1089023.1089026.

[44] Emmanuel J Candes and Terence Tao. Decoding by linear programming. *IEEE transactions on information theory*, 51(12):4203–4215, 2005.

[45] Clément L Canonne. A survey on distribution testing: Your data is big. but is it blue? *Theory of Computing*, pages 1–100, 2020.

[46] Anthony Carbery and James Wright. Distributional and l^q norm inequalities for polynomials over convex bodies in $\mathbb{R}^n$. *Mathematical research letters*, 8(3):233–248, 2001.

[47] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 267–284, 2019.

[48] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

[49] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914. IEEE, 2022.

[50] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. *International Conference on Learning Representations*, 2023.

[51] Nicolas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5253–5270, 2023.

[52] Timothy M Chan. Applications of chebyshev polynomials to low-dimensional computational geometry. *Journal of Computational Geometry*, 9(2):3–20, 2018.

[53] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[54] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11(4), 2010.

[55] Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 1032–1043, 2017.

[56] Moses Charikar and Erik Waingarten. The johnson-lindenstrauss lemma for clustering and subspace approximation: From coresets to dimension reduction. *CoRR*,

abs/2205.00371, 2022. doi:10.48550/arXiv.2205.00371. URL https://doi.org/10.48550/arXiv.2205.00371.

[57] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[58] Moses Charikar, Michael Kapralov, Navid Nouri, and Paris Siminelakis. Kernel density estimation through density constrained near neighbor search. *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 172–183, 2020.

[59] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[60] Dingfan Chen, Ning Yu, Yang Zhang, and Mario Fritz. Gan-leaks: A taxonomy of membership inference attacks against generative models. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 343–362, 2020.

[61] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[62] Xiaoyu Chen, Shaofeng H-C Jiang, and Robert Krauthgamer. Streaming euclidean max-cut: Dimension vs data reduction. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 170–182, 2023.

[63] Xue Chen and Eric Price. Condition number-free query and active learning of linear families. *CoRR*, abs/1711.10051, 2017.

[64] Yichen Chen, Yinyu Ye, and Mengdi Wang. Approximation hardness for A class of sparse optimization problems. *J. Mach. Learn. Res.*, 20:38:1–38:27, 2019.

[65] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[66] Andrew M. Childs, Shih-Han Hung, and Tongyang Li. Quantum query complexity with matrix-vector products. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 55:1–55:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.55. URL https://doi.org/10.4230/LIPIcs.ICALP.2021.55.

[67] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference, STOC*, pages 81–90, 2013.

[68] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation.

In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 163–172. ACM, 2015.

[69] Benjamin Coleman and Anshumali Shrivastava. A one-pass distributed and private sketch for kernel sums with applications to machine learning at scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3252–3265, 2021.

[70] Michael AA Cox and Trevor F Cox. Multidimensional scaling. In *Handbook of data visualization*, pages 315–347. Springer, 2008.

[71] Geoff Davis, Stephane Mallat, and Marco Avellaneda. Adaptive greedy approximations. *Constructive approximation*, 13(1):57–98, 1997.

[72] Soham De, Leonard Berrada, Jamie Hayes, Samuel L Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale. *arXiv preprint arXiv:2204.13650*, 2022.

[73] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[74] Ivan Dokmanic, Reza Parhizkar, Juri Ranieri, and Martin Vetterli. Euclidean distance matrices: essential theory, algorithms, and applications. *IEEE Signal Processing Magazine*, 32(6):12–30, 2015.

[75] David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52 (4):1289–1306, 2006.

[76] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 265–284. Springer, 2006.

[77] Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. Boosting and differential privacy. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 51–60. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.12. URL https://doi.org/10.1109/FOCS.2010.12.

[78] Yonina C Eldar and Gitta Kutyniok. *Compressed sensing: theory and applications*. Cambridge university press, 2012.

[79] Rogers Epstein and Sandeep Silwal. Property testing of lp-type problems. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[80] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[81] Dan Feldman, Amos Fiat, Haim Kaplan, and Kobbi Nissim. Private coresets. In

Proceedings of the forty-first annual ACM symposium on Theory of computing, pages 361–370, 2009.

[82] Dean P. Foster, Howard J. Karloff, and Justin Thaler. Variable selection is hard. In *Proceedings of The 28th Conference on Learning Theory, COLT*, volume 40, pages 696–709, 2015.

[83] Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Applied and Numerical Harmonic Analysis. Birkhäuser, 2013. ISBN 978-0-8176-4947-0. doi:10.1007/978-0-8176-4948-7. URL https://doi.org/10.1007/978-0-8176-4948-7.

[84] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015.

[85] Shmuel Friedland. Lower bounds for the first eigenvalue of certain m-matrices associated with graphs. *Linear Algebra and its Applications*, 172:71–84, 1992.

[86] Shmuel Friedland and Reinhard Nabben. On Cheeger-type inequalities for weighted graphs. *Journal of Graph Theory*, 41(1):1–17, 2002.

[87] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

[88] Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

[89] Alexander G Gray and Andrew W Moore. N-body problems in statistical learning. *Advances in neural information processing systems*, pages 521–527, 2001.

[90] Alexander G Gray and Andrew W Moore. Nonparametric density estimation: Toward computational tractability. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 203–211. SIAM, 2003.

[91] Anupam Gupta, Aaron Roth, and Jonathan R. Ullman. Iterative constructions and private data release. In Ronald Cramer, editor, *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*, pages 339–356. Springer, 2012. doi:10.1007/978-3-642-28914-9_19. URL https://doi.org/10.1007/978-3-642-28914-9_19.

[92] Aparna Gupte and Vinod Vaikuntanathan. The fine-grained hardness of sparse linear regression, 2021.

[93] Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. *Advances in Neural Information Processing Systems*, 35:22911–22924, 2022.

[94] Rob Hall, Alessandro Rinaldo, and Larry Wasserman. Differential privacy for functions and functional data. *The Journal of Machine Learning Research*, 14(1):703–727, 2013.

[95] Sariel Har-Peled, Piotr Indyk, and Sepideh Mahabadi. Approximate sparse linear regression. In *45th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 107, pages 77:1–77:14, 2018.

[96] Moritz Hardt and Kunal Talwar. On the geometry of differential privacy. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 705–714, 2010.

[97] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.

[98] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, 36(3):1171–1220, 2008.

[99] Liisa Holm and Chris Sander. Protein structure comparison by alignment of distance matrices. *Journal of molecular biology*, 233(1):123–138, 1993.

[100] Charlie Hou, Hongyuan Zhan, Akshat Shrivastava, Sid Wang, Aleksandr Livshits, Giulia Fanti, and Daniel Lazar. Privately customizing prefinetuning to better match user data in federated learning. In *ICLR 2023 Workshop on Pitfalls of limited data and computation for Trustworthy ML*, 2023.

[101] Nate Eldredge (https://math.stackexchange.com/users/822/nate eldredge). The lebesgue measure of zero set of a polynomial function is zero. Mathematics Stack Exchange. URL https://math.stackexchange.com/q/1920527. URL:https://math.stackexchange.com/q/1920527 (version: 2021-07-22).

[102] Zhiyi Huang and Aaron Roth. Exploiting metric structure for efficient private query release. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 523–534. SIAM, 2014. doi:10.1137/1.9781611973402.39. URL https://doi.org/10.1137/1.9781611973402.39.

[103] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[104] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4): 512–530, 2001.

[105] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

[106] Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 39–42. SIAM, 2007. URL http://dl.acm.org/citation.cfm?id=1283383.1283388.

[107] Piotr Indyk and Assaf Naor. Nearest-neighbor-preserving embeddings. *ACM Transactions on Algorithms (TALG)*, 3(3):31–es, 2007.

[108] Piotr Indyk and Sandeep Silwal. Faster linear algebra for distance matrices. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/e7599c4b309e39e444a7dcf92572fae1-Abstract-Conference.html.

[109] Piotr Indyk and Sandeep Silwal. Faster linear algebra for distance matrices. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 35576–35589. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/e7599c4b309e39e444a7dcf92572fae1-Paper-Conference.pdf.

[110] Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat. Closest pair problems in very high dimensions. In *International Colloquium on Automata, Languages, and Programming*, pages 782–792. Springer, 2004.

[111] Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Practical data-dependent metric compression with provable guarantees. *Advances in Neural Information Processing Systems*, 30, 2017.

[112] Piotr Indyk, Ali Vakilian, Tal Wagner, and David P. Woodruff. Sample-optimal low-rank approximation of distance matrices. In *Conference on Learning Theory, COLT*, pages 1723–1751, 2019.

[113] Pitor Indyk, Ali Vakilian, Tal Wagner, and David P Woodruff. Sample-optimal low-rank approximation of distance matrices. In *Conference on Learning Theory*, pages 1723–1751. PMLR, 2019.

[114] Zachary Izzo, Sandeep Silwal, and Samson Zhou. Dimensionality reduction for wasserstein barycenter. *Advances in neural information processing systems*, 34:15582–15594, 2021.

[115] Thathachar S Jayram and David P Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms (TALG)*, 9(3):1–17, 2013.

[116] Li Jiang, Shaoshuai Shi, Xiaojuan Qi, and Jiaya Jia. Gal: Geometric adversarial loss for single-view 3d-object reconstruction. In *Proceedings of the European conference on computer vision (ECCV)*, pages 802–816, 2018.

[117] Shaofeng H-C Jiang, Robert Krauthgamer, and Shay Sapir. Moderate dimension reduction for k-center clustering. *arXiv preprint arXiv:2312.01391*, 2023.

[118] Ce Jin and Yinzhan Xu. Shaving logs via large sieve inequality: Faster algorithms for sparse convolution and more, 2024.

[119] W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 01 1984. doi:10.1090/conm/026/737400.

[120] William B Johnson. Extensions of lipshitz mapping into hilbert space. In *Conference modern analysis and probability, 1984*, pages 189–206, 1984.

[121] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

[122] Matti Karppa, Martin Aumüller, and Rasmus Pagh. Deann: Speeding up kernel-density estimation using approximate nearest neighbor search. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, pages 3108–3137, 2022.

[123] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598, 2011.

[124] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[125] Joseph B Kruskal. *Multidimensional scaling*. Number 11. Sage, 1978.

[126] Marek Kuczma. *An introduction to the theory of functional equations and inequalities: Cauchy's equation and Jensen's inequality*. Springer Science & Business Media, 2009.

[127] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966. PMLR, 2015.

[128] Christiane Lammersen, Anastasios Sidiropoulos, and Christian Sohler. Streaming embeddings with slack. In Frank K. H. A. Dehne, Marina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Algorithms and Data Structures, 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings*, volume 5664 of *Lecture Notes in Computer Science*, pages 483–494. Springer, 2009. doi:10.1007/978-3-642-03367-4_42. URL https://doi.org/10.1007/978-3-642-03367-4_42.

[129] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. 1950.

[130] Kasper Green Larsen and Jelani Nelson. The johnson-lindenstrauss lemma is optimal for linear dimensionality reduction. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 82:1–82:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ICALP.2016.82. URL https://doi.org/10.4230/LIPIcs.ICALP.2016.82.

[131] Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 633–638. IEEE, 2017.

[132] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[133] Dongryeol Lee and Alexander Gray. Fast high-dimensional kernel summations using the monte carlo multipole method. *Advances in Neural Information Processing Systems*, 21:929–936, 2008.

[134] Dongryeol Lee, Andrew W Moore, and Alexander G Gray. Dual-tree fast gauss transforms. In *Advances in Neural Information Processing Systems*, pages 747–754, 2006.

[135] Troy Lee, Miklos Santha, and Shengyu Zhang. Quantum algorithms for graph problems with cut queries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 939–958. SIAM, 2021.

[136] Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. *SIAM Journal on Computing*, 47(6):2315–2336, 2018.

[137] Chun-Liang Li, Tomas Simon, Jason Saragih, Barnabás Póczos, and Yaser Sheikh. Lbs autoencoder: Self-supervised fitting of articulated meshes to point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11967–11976, 2019.

[138] Gen Li, Xingyu Xu, and Yuantao Gu. Lower bound for rip constants and concentration of sum of top order statistics. *IEEE Transactions on Signal Processing*, 68:3169–3178, 2020.

[139] Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. Large language models can be strong differentially private learners. In *International Conference on Learning Representations*, 2021.

[140] Yi Li, Huy L Nguyen, and David P Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 174–183, 2014.

[141] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pages 464–483, 2020.

[142] Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, Harsha Nori, and Sergey Yekhanin. Differentially private synthetic data via foundation model APIs 1: Images. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=YEhQs8POIo.

[143] Xinyu Luo, Christopher Musco, and Cas Widdershoven. Dimensionality reduction for general kde mode finding. In *International Conference on Machine Learning*. PMLR, 2023.

[144] Sepideh Mahabadi. Approximate nearest line search in high dimensions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 337–354, 2015.

[145] Tung Mai, Alexander Munteanu, Cameron Musco, Anup Rao, Chris Schwiegelshohn, and David Woodruff. Optimal sketching bounds for sparse linear regression. In *International Conference on Artificial Intelligence and Statistics*, pages 11288–11316. PMLR, 2023.

[146] Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In Moses

Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1027–1038. ACM, 2019. doi:10.1145/3313276.3316350. URL https://doi.org/10.1145/3313276.3316350.

[147] William B March, Bo Xiao, and George Biros. Askit: Approximate skeletonization kernel-independent treecode in high dimensions. *SIAM Journal on Scientific Computing*, 37(2):A1089–A1110, 2015.

[148] Jiri Matousek. *Lectures on discrete geometry*, volume 212. Springer Science & Business Media, 2013.

[149] Russell Merris. Laplacian matrices of graphs: a survey. *Linear algebra and its applications*, 197:143–176, 1994.

[150] Raphael A Meyer, Cameron Musco, Christopher Musco, and David P Woodruff. Hutch++: Optimal stochastic trace estimation. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 142–155. SIAM, 2021.

[151] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[152] Vlad I Morariu, Balaji Vasan Srinivasan, Vikas C Raykar, Ramani Duraiswami, and Larry S Davis. Automatic online tuning for fast gaussian summation. In *NIPS*, pages 1113–1120, 2008.

[153] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, pages 1396–1404, 2015.

[154] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Advances in neural information processing systems*, 28, 2015.

[155] Cameron Musco and David P Woodruff. Sublinear time low-rank approximation of positive semidefinite matrices. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 672–683. IEEE, 2017.

[156] Shyam Narayanan, Sandeep Silwal, Piotr Indyk, and Or Zamir. Randomized dimensionality reduction for facility location and single-linkage clustering. In *International Conference on Machine Learning*, pages 7948–7957. PMLR, 2021.

[157] Balas K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–234, 1995.

[158] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[159] Jeff M Phillips. ε-samples for kernels. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1622–1632. SIAM, 2013.

[160] Jeff M. Phillips and Wai Ming Tai. Improved coresets for kernel density estimates. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2718–2727, 2018.

[161] Jeff M. Phillips and Wai Ming Tai. Near-optimal coresets of kernel density estimates. *Discret. Comput. Geom.*, 63(4):867–887, 2020. doi:10.1007/s00454-019-00134-6. URL https://doi.org/10.1007/s00454-019-00134-6.

[162] Jeff M Phillips and Wai Ming Tai. Near-optimal coresets of kernel density estimates. *Discrete & Computational Geometry*, 63(4):867–887, 2020.

[163] Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Guha Thakurta. How to dp-fy ml: A practical guide to machine learning with differential privacy. *Journal of Artificial Intelligence Research*, 77:1113–1201, 2023.

[164] Eric Price and David P Woodruff. (1+ eps)-approximate sparse recovery. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 295–304. IEEE, 2011.

[165] Eric Price, Sandeep Silwal, and Samson Zhou. Hardness and algorithms for robust and sparse optimization. In *International Conference on Machine Learning*, pages 17926–17944. PMLR, 2022.

[166] Kent Quanrud. Spectral sparsification of metrics and kernels. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1445–1464, 2021.

[167] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[168] Parikshit Ram, Dongryeol Lee, William March, and Alexander Gray. Linear-time algorithms for pairwise statistical problems. *Advances in Neural Information Processing Systems*, 22:1527–1535, 2009.

[169] Cyrus Rashtchian, David P. Woodruff, and Hanlin Zhu. Vector-matrix-vector queries for solving linear algebra, statistics, and graph problems. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 26:1–26:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.APPROX/RANDOM.2020.26. URL https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2020.26.

[170] Dhruv Rohatgi. Conditional hardness of earth mover distance. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[171] Sushant Sachdeva and Nisheeth K. Vishnoi. Faster algorithms via approximation theory. *Found. Trends Theor. Comput. Sci.*, 9(2):125–210, 2014. doi:10.1561/0400000065. URL https://doi.org/10.1561/0400000065.

[172] Sushant Sachdeva, Nisheeth K Vishnoi, et al. Faster algorithms via approximation theory. *Foundations and Trends® in Theoretical Computer Science*, 9(2):125–210, 2014.

[173] Bernhard Schölkopf, Alexander J Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[174] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[175] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[176] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 177–189. PMLR, 2022.

[177] Paris Siminelakis, Kexin Rong, Peter Bailis, Moses Charikar, and Philip Alexander Levis. Rehashing kernel evaluation in high dimensions. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, pages 5789–5798, 2019.

[178] Vikrant Singhal and Thomas Steinke. Privately learning subspaces. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 1312–1324, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/09b69adcd7cbae914c6204984097d2da-Abstract.html.

[179] Anthony Man-Cho So and Yinyu Ye. Theory of semidefinite programming for sensor network localization. *Mathematical Programming*, 109(2):367–384, 2007.

[180] Zhao Song, David Woodruff, Zheng Yu, and Lichen Zhang. Fast sketching of polynomial kernels of polynomial degree. In *International Conference on Machine Learning*, pages 9812–9823. PMLR, 2021.

[181] Daniel A Spielman. The laplacian matrices of graphs. In *Plenary Talk, IEEE Intern. Symp. Inf. Theory (ISIT)n*, 2016.

[182] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[183] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90, 2004.

[184] Erik B Sudderth, Michael I Mandel, William T Freeman, and Alan S Willsky. Visual

hand tracking using nonparametric belief propagation. In *2004 Conference on Computer Vision and Pattern Recognition Workshop*, pages 189–189. IEEE, 2004.

[185] Xiaoming Sun, David P Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. *ACM Transactions on Algorithms (TALG)*, 17(4): 1–19, 2021.

[186] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[187] Wai Ming Tai. Optimal coreset for gaussian kernel density estimation. In *38th International Symposium on Computational Geometry, SoCG*, pages 63:1–63:15, 2022.

[188] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[189] Florian Tramèr, Reza Shokri, Ayrton San Joaquin, Hoang Le, Matthew Jagielski, Sanghyun Hong, and Nicholas Carlini. Truth serum: Poisoning machine learning models to reveal their secrets. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2779–2792, 2022.

[190] Salil Vadhan. *The Complexity of Differential Privacy*, pages 347–450. Springer, Yehuda Lindell, ed., 2017. URL https://link.springer.com/chapter/10.1007/978-3-319-57048-8_7.

[191] Salil Vadhan. *The Complexity of Differential Privacy*, pages 347–450. Springer, Yehuda Lindell, ed., 2017. URL https://link.springer.com/chapter/10.1007/978-3-319-57048-8_7.

[192] Tal Wagner, Yonatan Naamad, and Nina Misrha. Fast private kernel density estimation via locality sensitive quantization. In *International Conference on Machine Learning*. PMLR, 2023.

[193] Ziyu Wan, Dongdong Chen, Yan Li, Xingguang Yan, Junge Zhang, Yizhou Yu, and Jing Liao. Transductive zero-shot learning with visual structure constraint. *Advances in neural information processing systems*, 32, 2019.

[194] Ziteng Wang, Chi Jin, Kai Fan, Jiaqi Zhang, Junliang Huang, Yiqiao Zhong, and Liwei Wang. Differentially private data releasing for smooth queries. *The Journal of Machine Learning Research*, 17(1):1779–1820, 2016.

[195] Kilian Q Weinberger and Lawrence K Saul. Unsupervised learning of image manifolds by semidefinite programming. *International journal of computer vision*, 70(1):77–90, 2006.

[196] Douglas Wiedemann. Solving sparse linear equations over finite fields. *IEEE transactions on information theory*, 32(1):54–62, 1986.

[197] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

[198] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pages 3447–3487. World Scientific, 2018.

[199] David Woodruff and Amir Zandieh. Near input sparsity time kernel embeddings via adaptive sampling. In *International Conference on Machine Learning*, pages 10324–10333. PMLR, 2020.

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

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

[202] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 458–472, 2022.

[203] Da Yu, Huishuai Zhang, Wei Chen, and Tie-Yan Liu. Do not let privacy overbill utility: Gradient embedding perturbation for private learning. In *International Conference on Learning Representations*, 2020.

[204] Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A. Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, Sergey Yekhanin, and Huishuai Zhang. Differentially private fine-tuning of language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=Q42f0dfjECO.

[205] Da Yu, Sivakanth Gopi, Janardhan Kulkarni, Zinan Lin, Saurabh Naik, Tomasz Lukasz Religa, Jian Yin, and Huishuai Zhang. Selective pre-training for private fine-tuning. *arXiv preprint arXiv:2305.13865*, 2023.

[206] Xiang Yue, Huseyin A. Inan, Xuechen Li, Girish Kumar, Julia McAnallen, Hoda Shajari, Huan Sun, David Levitan, and Robert Sim. Synthetic text generation with differential privacy: A simple and practical recipe. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 1321–1342. Association for Computational Linguistics, 2023. doi:10.18653/V1/2023.ACL-LONG.74. URL https://doi.org/10.18653/v1/2023.acl-long.74.

[207] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016. URL http://www.bmva.org/bmvc/2016/papers/paper087/index.html.

[208] Yan Zheng, Jeffrey Jestes, Jeff M Phillips, and Feifei Li. Quality and efficiency for

kernel density estimates in large data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 433–444, 2013.

[209] Zeyuan Allen Zhu, Rati Gelashvili, and Ilya P. Razenshteyn. Restricted isometry property for general p-norms. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, volume 34 of *LIPIcs*, pages 451–460. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.SOCG.2015.451. URL https://doi.org/10.4230/LIPIcs.SOCG.2015.451.