ONe Index for All Kernels (ONIAK): A Zero Re-Indexing LSH Solution to ANNS-ALT (After Linear Transformation)

Jingfan Meng[†] Huayi Wang[†] Jun Xu[†] Mitsunori Ogihara[‡]
[†]{jmeng40, hwang762}@gatech.edu, jx@cc.gatech.edu, [‡]ogihara@cs.miami.edu
[†]Georgia Institute of Technology, Atlanta, USA [‡]University of Miami, Coral Gables, USA

ABSTRACT

In this work, we formulate and solve a new type of approximate nearest neighbor search (ANNS) problems called ANNS after linear transformation (ALT). In ANNS-ALT, we search for the vector (in a dataset) that, after being linearly transformed by a user-specified query matrix, is closest to a query vector. It is a very general mother problem in the sense that a wide range of baby ANNS problems that have important applications in databases and machine learning can be reduced to and solved as ANNS-ALT, or its dual that we call ANNS-ALTD. We propose a novel and computationally efficient solution, called ONe Index for All Kernels (ONIAK), to ANNS-ALT and all its baby problems when the data dimension d is not too large (say $d \leq 200$). In ONIAK, a universal index is built, once and for all, for answering all future ANNS-ALT queries that can have *distinct* query matrices. We show by experiments that, when d is not too large, ONIAK has better query performance than linear scan on the mother problem (of ANNS-ALT), and has query performances comparable to those of the state-of-the-art solutions on the baby problems. However, the algorithmic technique behind this universal index approach suffers from a so-called dimension blowup problem that can make the indexing time prohibitively long for a large dataset. We propose a novel algorithmic technique, called fast GOE quadratic form (FGoeQF), that completely solves the (prohibitively long indexing time) fallout of the dimension blowup problem. We also propose a Johnson-Lindenstrauss transform (JLT) based ANNS-ALT (and ANNS-ALTD) solution that significantly outperforms any competitor when d is large.

PVLDB Reference Format:
Jingfan Meng, Huayi Wang, Jun Xu, Mitsunori Ogihara. ONe Index for All Kernels (ONIAK): A Zero Re-Indexing LSH Solution to ANNS-ALT (After Linear Transformation) . PVLDB, 15(13): 3937 - 3949, 2022.
doi:10.14778/3565838.3565847

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Jeffery-Meng/ONIAK.

1 INTRODUCTION

Approximate nearest neighbor search (ANNS) is a fundamental algorithmic problem with numerous applications in many areas of computer science, especially databases [10, 32] and machine learning [14, 39]. In a standard ANNS problem, we answer ANNS queries over a large dataset $\mathcal{D}$ that lies in a high-dimensional space. Given a query $\vec{q}$ that lies in the *same space*, we need to find one or more points in $\mathcal{D}$ that are closest to $\vec{q}$ in a certain distance metric such as L_2 (the Euclidean distance).

In this work, we attack a fundamentally different type of ANNS problems. In this type, like in other ANNS problems, we need to answer queries over a dataset $\mathcal{D}$ that lies in a d-dimensional space. However, unlike in other ANNS problems, the queries lie in a very different space than $\mathcal{D}$ in the following way. For $i = 1, 2, \cdots$, the i^{th} query is a tuple $(M_i, \vec{q}_i)$, where M_i is an $r_i \times d$ *query matrix* and $\vec{q}_i$ is an r_i-dimensional *query vector*. Given this query $(M_i, \vec{q}_i)$, the goal is to find $\vec{x} \in \mathcal{D}$ such that the vector $M_i\vec{x}$ is the closest to $\vec{q}_i$ in L_2. Formally, we seek to find $\arg\min_{\vec{x}\in\mathcal{D}} \|M_i\vec{x} - \vec{q}_i\|_2$. For $i = 1, 2, \cdots$, the values of the matrices M_i, the vectors $\vec{q}_i$, and even the values of parameters r_i can generally be different. We call this problem ANNS after linear transformation, or ANNS-ALT for short, because we are looking for $\vec{x} \in \mathcal{D}$ that, when linearly transformed by the query matrix M_i, is closest to the query vector $\vec{q}_i$.

We adopt the following notational convention throughout this paper. We use a lower-case letter with a rightward arrow on top of it, such as $\vec{x}$, to denote a column vector, and use its transpose, such as $\vec{x}^T$, to denote a row vector. We use the same letter without an arrow on the top but with a subscript, such as x_i, to denote a scalar in it. Also, we use a capital letter, such as A, to denote a matrix, and a corresponding lower case letter with two subscripts, such as $a_{i,j}$, to denote a scalar in it.

1.1 ANNS-ALT: An Important Mother Problem

As we will show in §4, the ANNS-ALT problem is the "mother" of several "baby" ANNS problems in the sense that every baby problem can be reduced to and solved as an ANNS-ALT problem. Although these baby problems have important applications in databases [32], machine learning [39], and computer vision [10], so far most of them do not have computationally efficient solutions. We motivate the ANNS-ALT problem using one such baby problem that is known as ANNS in Mahalanobis distance, or ANNS-M for short. The Mahalanobis distance between any two d-dimensional points $\vec{x}$ and $\vec{y}$, denoted as $D_M^\Sigma(\vec{x}, \vec{y})$, is defined as $D_M^\Sigma(\vec{x}, \vec{y}) \triangleq \sqrt{(\vec{x} - \vec{y})^T\Sigma(\vec{x} - \vec{y})}$, where Σ is a $d\times d$ positive semidefinite matrix which we call a Mahalanobis *kernel*. In ANNS-M, each query is a tuple $(\Sigma_i, \vec{q}_i)$, where Σ_i is such a $d \times d$ kernel matrix that defines the Mahalanobis distance, and $\vec{q}_i$ is a query vector. The goal is to find $\vec{x} \in \mathcal{D}$ that is nearest to $\vec{q}_i$ in $D_M^{\Sigma_i}$. In general, both the matrix Σ_i and the vector $\vec{q}_i$ can change from one query to another.

Proceedings of the VLDB Endowment, Vol. 15, No. 13 ISSN 2150-8097.
doi:10.14778/3565838.3565847

If the kernel matrices in all ANNS-M queries were identical and known in advance, then this special case, which we call static (fixed-kernel) ANNS-M, can be reduced to and solved as a much easier ANNS-L_2 problem as follows. Let the rank of the $d \times d$ Mahalanobis kernel matrix Σ be r ($r \leq d$). Since Σ is positive semidefinite, there exists an $r \times d$ matrix U such that $\Sigma = U^T U$ [38]. Hence, we have $D_M^{\Sigma}(\vec{x}, \vec{y}) = \sqrt{(\vec{x}-\vec{y})^T \Sigma (\vec{x}-\vec{y})} = \sqrt{(\vec{x}-\vec{y})^T U^T U (\vec{x}-\vec{y})} = \|U(\vec{x}-\vec{y})\|_2 = \|U\vec{x} - U\vec{y}\|_2$. This relationship implies that, if the vector $\vec{x}^*$ is the closest to a query vector $\vec{q}$ in Mahalanobis distance D_M^{Σ} among vectors in $\mathcal{D}$, then the mapped (by U) vector $U\vec{x}^*$ is the closest to the mapped query vector $U\vec{q}$ in L_2 distance among vectors in $U(\mathcal{D}) \triangleq \{U\vec{x} \mid \vec{x} \in \mathcal{D}\}$, and vice versa. In other words, static ANNS-M is equivalent to the ANNS-L_2 problem of searching for the nearest neighbor of $U\vec{q}$ in $U(\mathcal{D})$.

In comparison, general ANNS-M, in which the kernel Σ can change from one query to another, is a much harder problem. For one thing, the naive approach of simply using the static ANNS-M solution above for the general ANNS-M problem, called *Rebuild LSH* [39], has a poor query performance as follows. In Rebuild LSH, every time the kernel changes, say from $\Sigma_i = U_i^T U_i$ to $\Sigma_{i+1} = U_{i+1}^T U_{i+1}$, we need to *re-project* $\mathcal{D}$ to $U_{i+1}(\mathcal{D})$ and "*re-hash*" $U_{i+1}(\mathcal{D})$, before the new query $(\Sigma_{i+1}, \vec{q}_{i+1})$ can be answered, since the old index for $U_i(\mathcal{D})$ no longer works for the new query matrix Σ_{i+1}. The re-indexing (including both re-projection and "re-hashing") process is very time-consuming: It takes slightly longer than processing an ANNS-M query using linear scan.

We reduce ANNS-M to, and solve it as, the following ANNS-ALT problem. Given any ANNS-M query $(\Sigma_i, \vec{q}_i)$ over $\mathcal{D}$, we convert it to the ANNS-ALT query $(U_i, U_i\vec{q}_i)$ also over $\mathcal{D}$, where $\Sigma = U_i^T U_i$. These two queries are equivalent because $D_M^{\Sigma}(\vec{x}, \vec{q}) = \|U_i\vec{x} - U_i\vec{q}\|_2$. Hence, our solution to ANNS-ALT, to be described next, also solves ANNS-M. We name this solution ONIAK (ONe Index for All Kernels), since when using it to solve ANNS-M, we need only to build, once and for all, a *universal index* for $\mathcal{D}$ that can answer all future queries with different kernels. Hence, ONIAK completely eliminates any need for re-indexing. The acronym ONIAK is inspired by a character named "the Great Oniak" in a 1960s sci-fi TV series titled *Lost in Space*. It is an appropriate name for this work, since an ANNS solution is all about avoiding the fate of having a nearest neighbor "lost" in a high-dimensional space.

1.2 ONIAK: A Universal Index for ANNS-ALT

Now we introduce the ONIAK approach of building a universal index on $\mathcal{D}$ that can efficiently answer an arbitrary sequence of ANNS-ALT queries $(M_i, \vec{q}_i)$, for $i = 1, 2, \cdots$. With the understanding that both M_i and $\vec{q}_i$ can change from one query to another, we drop the subscript i from both and focus on how ONIAK processes a single query $(M, \vec{q})$ in the sequel. Recall that the query objective is to find $\vec{x} \in \mathcal{D}$ that minimizes $\|M\vec{x} - \vec{q}\|_2$. As to be elaborated in §2.5, it suffices for ONIAK to solve the following simpler but equivalent form of the ANNS-ALT problem, which we name ANNS-SALT (S stands for simple). In ANNS-SALT, each query is just a matrix M, and the objective is to find $\vec{x} \in \mathcal{D}$ that minimizes $\|M\vec{x}\|_2$.

Given an ANNS-SALT query M, ONIAK converts it to an ANNS-L_2 query using a technique pioneered in a computer vision paper [10] for solving one baby problem of ANNS-ALT. This technique is an *asymmetric mappings pair* (AMP) $f(\cdot)$ and $g(\cdot)$ that has the following *order preservation property*: Given any matrix M and any two vectors $\vec{x}_1, \vec{x}_2$, we have $\|f(\vec{x}_1) - g(M)\|_2 \leq \|f(\vec{x}_2) - g(M)\|_2$ if and only if $\|M\vec{x}_1\|_2 \leq \|M\vec{x}_2\|_2$. As a result, the ANNS-SALT problem of finding $\vec{x}$ in $\mathcal{D}$ that minimizes $\|M\vec{x}\|_2$ is equivalent to the ANNS-L_2 problem of finding $\vec{y}$ in $f(\mathcal{D}) \triangleq \{f(\vec{x}) \mid \vec{x} \in \mathcal{D}\}$ that is closest to the vector $g(M)$. In ONIAK, when the dimension d is not too large (say $d \leq 200$), this ANNS-L_2 problem is then solved using the state-of-the-art locality sensitive hashing (LSH) scheme called Gaussian Projection LSH (GP-LSH) [12].

Unfortunately, this AMP technique suffers from the following *dimension blowup* problem that severely limits the efficacy of any ANNS solution based on this technique, when the dimension d of the dataset $\mathcal{D}$ becomes large: $f(\cdot)$ maps a d-dimensional vector $\vec{x}$ to a d^2-dimensional vector $f(\vec{x})$. Hence in the aforementioned ANNS-L_2 problem that ANNS-SALT reduces to, each mapped (by $f(\cdot)$) vector in the dataset $f(\mathcal{D})$ is of d^2 dimensions. As a result, to evaluate an LSH function $h(\cdot)$ on such a mapped vector, say $f(\vec{x})$, conceivably requires at least $O(d^2)$ time, since just to read the input (argument to $h(\cdot)$) $f(\vec{x})$ takes $O(d^2)$ time. This $O(d^2)$ time complexity leads to an unacceptably long indexing time for building the aforementioned universal index for the mapped dataset $f(\mathcal{D})$, since in the indexing phase, we need to evaluate hundreds to thousands of different LSH functions on each mapped data vector $f(\vec{x}) \in f(\mathcal{D})$. Likely for this limitation, the entire solution approach based on this technique has all but been abandoned after the hyperplane hashing work [32]. A major contribution of our work is an algorithmic breakthrough, to be introduced next, that completely solves the fallout of this dimension blowup problem and as a result rescues this approach from near-certain demise in the era of big data.

This breakthrough, called *fast Gaussian orthogonal ensemble quadratic form* (FGoeQF) and to be described in §2.2, can reduce the time complexity of computing a hash value from $O(d^2)$ to $O(d \log d)$, when the LSH function family used is the aforementioned GP-LSH. This $O(d/\log d)$-fold reduction in indexing time is clearly "life-saving" to all existing ANNS solutions that suffer from this dimension blowup problem, including [10, 16, 32]. In addition, we recently discover that FGoeQF can reduce the indexing time of the state-of-the-art solution, called locality sensitive teaching [35], to an emerging machine learning problem known as iterative machine teaching [22], in which the dimension blowup problem stems from a standard stochastic gradient descent (SGD) procedure rather than from AMP. This suggests that the dimension blowup is a fundamental and prevalent problem, and that our FGoeQF solution may possibly benefit many more future applications in database and machine learning .

1.3 JLT-Based Solution to ANNS-ALT

In addition to the dimension blowup problem, we need to deal with a fundamental limitation of AMP. We will show in §2.4 that considerable noise is introduced in this AMP process and the resulting maximum signal-to-noise ratio (SNR) scales as $O(1/d)$. As a result, when $d > 200$, ONIAK no longer outperforms linear scan much according to our experiments. We propose a different, and surprisingly effective, ANNS-ALT solution for the case of large d. This solution, which is based on a well-known algorithmic technique for

dimensionality reduction called Johnson-Lindenstrauss transform (JLT) [2], works as follows. At initialization, we generate and fix an $l \times d$ Gaussian random matrix Θ, where $l \ll d$. Then given a $d \times d$ ANNS-ALT query matrix (into which an $r \times d$ query matrix with $r < d$ can be zero-padded) M and query vector $\vec{q}$, we first compute the matrix product $M' = \Theta M$, for which we pay a one-time computational complexity of $O(ld^2)$, and $\vec{q'} = \Theta\vec{q}$ (for a cost of $O(d^2)$). This matrix multiplication (projection) is called a JLT, and the resulting M' is an $l \times d$ matrix. Then for *every* ANNS-SALT candidate $\vec{x} \in \mathcal{D}$, we compute $M'\vec{x} - \vec{q'}$ using $O(ld)$ time, instead of $M\vec{x} - \vec{q}$ using $O(d^2)$ time. Since $\|M'\vec{x} - \vec{q'}\|_2 = \|\Theta(M\vec{x} - \vec{q})\|_2 \approx \|M\vec{x} - \vec{q}\|_2$ due to the property of the JLT [2], we can filter out most of the unpromising candidates according to their $\|M'\vec{x} - \vec{q'}\|_2$ values, so that the expensive $M\vec{x} - \vec{q}$ (ground truth) computation only needs to be performed for promising candidates. We refer to this filtering-based (by JLT) solution as JLT in the sequel. As we will show, JLT is much faster than linear scan (LS), and this outperformance grows roughly as $O(d)$. For example, when $d = 4096$, the query time of JLT is roughly 388 times shorter than that of LS. Another salient property of JLT is that it does not require an index (and hence has an index size of 0). When d is small (say $d < 64$), JLT cannot outperform LS as there is "no more dimension to reduce", whereas ONIAK performs fairly well in this case. Hence our solutions JLT and ONIAK complement each other well.

Although JLT is a well-established technique, we claim JLT as our contribution for three reasons. First, JLT has never been mentioned as a possible solution to any baby problem of ANNS-ALT [9, 32, 39], even though it can significantly outperform the state-of-the-art solutions to two such baby problems as we will elaborate in § 5 and §6. Second, this work provides the first evaluation study on the efficacy of JLT in the ANNS-ALT contexts. Third, we propose to combine ONIAK with JLT when d is moderately large (say between 64 and 200). ONIAK+JLT outperforms JLT by up to 1.6 times, as shown in §6.

The contributions of this work are summarized as follows. First, we formulate ANNS-ALT (and its dual problem), and show that it is the mother of a wide range of ANNS problems (§4). Second, we propose the ONIAK solution (§2), to ANNS-ALT, that eliminates any need for re-indexing after the initial (universal) indexing phase, for d that is not too large. Third, we propose the JLT solution for large d that can significantly outperform any competitor.

Last, we propose the FGoeQF technique (§3) that solves the fallout of the dimension blowup problem in a lossless manner.

2 THE ONIAK SOLUTION TO ANNS-ALT

In this section, we describe the ONIAK solution (except for FGoeQF) to ANNS-SALT, the aforementioned simple yet equivalent (to be established in §2.5) form of ANNS-ALT. We first formulate ANNS-SALT and then describe the aforementioned AMP technique that reduces ANNS-SALT to ANNS-L_2, in §2.1. This reduction causes the aforementioned dimension blowup problem that we will introduce in §2.2. We describe the aforementioned low SNR issue of ONIAK in §2.4 and the possible external-memory implementation of ONIAK in §2.6.

2.1 Asymmetric Mappings Pair (AMP)

As described earlier, each ANNS-SALT query is just a query matrix M, and the goal is to find $\vec{x} \in \mathcal{D}$ that minimizes $\|M\vec{x}\|_2$. In ANNS-SALT, the dataset $\mathcal{D}$ needs to be *normalized* in the sense that every $\vec{x} \in \mathcal{D}$ has the same L_2 norm. The proof of the equivalence between ANNS-ALT and ANNS-SALT can be found in §2.5.

The ANNS-SALT problem generalizes a well-known ANNS problem called *approximate hyperplane-to-point searching* (AH2PS) [32], and the AMP used in ONIAK for solving ANNS-SALT generalizes the AMP used in [32] for solving AH2PS. In AH2PS, the dataset $\mathcal{D}$ needs to be normalized like in ANNS-SALT. Each query in it is a d-dimensional vector $\vec{a}$, and the query objective is to find $\vec{x} \in \mathcal{D}$ that minimizes $|\vec{a}^T\vec{x}|$. ANNS-SALT degenerates to AH2PS when the query matrix M becomes a $1 \times d$ row vector $\vec{a}^T$. In ONIAK, the following AMP is used, where $\text{vec}(\cdot)$ denotes flattening a matrix in the row-major order into a vector.

$$f(\vec{x}) \triangleq \text{vec}(\vec{x}\vec{x}^T), \qquad g(M) \triangleq -\text{vec}(M^TM). \tag{1}$$

This AMP reduces the ANNS-SALT problem to the ANNS-L_2 problem of searching for the nearest neighbor of $g(M) = -\text{vec}(M^TM)$ in $f(\mathcal{D})$, thanks to the following order preservation property: For any two d-dimensional data vectors $\vec{x}_1, \vec{x}_2$ in $\mathcal{D}$ and any $r \times d$ query matrix M, we have $\|f(\vec{x}_1) - g(M)\|_2 \le \|f(\vec{x}_2) - g(M)\|_2$ if and only if $\|M\vec{x}_1\|_2 \le \|M\vec{x}_2\|_2$. This order preservation property is a corollary of Theorem 2.1: The LHS of (2) becomes larger when and only when $\|M\vec{x}\|_2$ (and hence the term $2\|M\vec{x}\|_2^2$) on the RHS becomes larger, since the other two terms on the RHS, namely $\|\vec{x}\|_2^4$ and $\|M^TM\|_F^2$, are constants (as $\mathcal{D}$ is normalized and M is a given query matrix).

Theorem 2.1 ([10]). *For any $(r \times d)$ query matrix M and any query vector $\vec{x} \in \mathbb{R}^d$, it holds that*

$$\|f(\vec{x}) - g(M)\|_2^2 = \|\vec{x}\|_2^4 + 2\|M\vec{x}\|_2^2 + \|M^TM\|_F^2, \tag{2}$$

where $\|\cdot\|_F$ is the Frobenius norm *defined in Definition 2.2.*

Definition 2.2. The Frobenius norm of a matrix M, denoted as $\|M\|_F$, is defined as $\sqrt{\sum_{i,j} m_{i,j}^2}$, the square root of the sum of the squares of all scalars in M.

2.2 The Dimension Blowup Problem

The dimension blowup problem is that, this AMP reduces an ANNS-SALT problem in which every data $\vec{x} \in \mathcal{D}$ is d-dimensional, to an ANNS-L_2 problem in which every (mapped) data vector $f(\vec{x}) = \text{vec}(\vec{x}\vec{x}^T)$ is d^2-dimensional. When d is large, this blowup can have a devastating impact on the indexing time as we will elaborate next, if an LSH scheme is used to solve the resulting d^2-dimensional ANNS-L_2 problem.

Suppose an LSH scheme is used to solve the ANNS-L_2 problem of searching in $f(\mathcal{D})$ for the nearest neighbor of $g(M) = -\text{vec}(M^TM)$, and to this end we have fixed a set, typically hundreds in cardinality, of such LSH functions. To build the index (LSH hash tables) for $f(\mathcal{D})$, we need to evaluate every such LSH function on every d^2-dimensional mapped data vector in $f(\mathcal{D})$. Let $h(\cdot)$ be one such LSH function. Evaluating $h(\cdot)$ on a mapped data vector $f(\vec{x}) = \text{vec}(\vec{x}\vec{x}^T)$ conceivably takes at least $O(d^2)$ time, since just to read the input $f(\vec{x})$ takes that long. As a result, the indexing time is intolerably

long for a large high-dimensional dataset as we will elaborate in §6.3. In comparison, in most other LSH-based ANNS schemes, this time complexity (of evaluating each LSH function) is only $O(d)$.

This $d \rightarrow d^2$ blowup problem, with its devastating effect on the indexing time, has been considered a vexing but unavoidable side effect of this AMP ever since its invention in [10]. A standard mitigation measure proposed in the literature [32] to this problem is to sample a certain percentage (say α) of coordinates (dimensions) and compute the LSH value based on the sampled coordinates. However, to bring this time complexity down to a "comfortable" level of $O(d)$, the sampling rate has to be as small as $O(1/d)$, but in this case the error between the computed and actual LSH values is unacceptably high when d is large. As the dimensions of datasets steadily increase over time, this dimensional blowup problem has made this AMP increasingly unappealing [41], so much so that it has all but been abandoned after [32].

We discover that this dimension blowup problem also arises in a different (than ANNS-SALT) application called locality sensitive teaching (LST) [36], for a different reason. The core idea of LST is to accelerate the stochastic gradient descent (SGD) [22] computation, that lies at the heart of modern machine learning, using LSH. It was shown in [36] that, when the linear regression model and the quadratic loss function are used, each SGD step therein reduces to the problem of finding a d-dimensional vector $\vec{x}$ in a (teaching example) dataset $\in \mathcal{D}$ such that the d^2-dimensional vector $\text{vec}(\vec{x}\vec{x}^T)$ is, with a decent probability (i.e., stochastically), close to a d^2-dimensional query vector in L_2 distance. The $\vec{x}$-to-$\text{vec}(\vec{x}\vec{x}^T)$ blowup in this LST solution results naturally from taking first derivative on the quadratic loss function, and hence has nothing to do with this or any AMP. From our understanding, unlike in ONIAK, nothing would prevent LST [36] from properly doing its job when d is large. In [36], this dimension blowup was mitigated using the aforementioned technique of sampling a small percentage of coordinates.

2.3 FGoeQF for Fast GP-LSH Indexing

We propose a much better solution to this blowup problem in both ONIAK and the LST solution [36]. This solution, called FGoeQF (Fast GOE Quadratic Form) and to be described in §3, brings the time complexity of hashing such a d^2-dimensional vector $\text{vec}(\vec{x}\vec{x}^T)$ from $O(d^2)$ down to $O(d \log d)$ without causing any accuracy loss, when the LSH function family used is GP-LSH [12]. Given an input vector $\vec{y}$, a GP-LSH function $h(\vec{y})$ is defined as $\lfloor (\eta(\vec{y})+b)/W \rfloor$, where $\eta(\cdot)$ is called the *raw hash function*, $W > 0$ is a constant, and b is a random variable (RV) (fixed after being generated) uniformly distributed in $[0, W]$. Here $\eta(\vec{y}) \triangleq \vec{\theta}^T\vec{y}$ is the *raw hash value*, where $\vec{\theta}$ (fixed after being generated during the initialization) is a *Gaussian random vector* defined as one that is comprised of i.i.d. standard Gaussian RVs. Clearly, computing $\eta(\vec{y})$ accounts solely for the $O(d^2)$ time complexity of computing $h(\vec{y})$, since the other three steps (plus, divide, and floor) have $O(1)$ time complexity.

It is not hard to verify that, for any $d \times d$ matrix M, we have $\text{vec}(M)^T\text{vec}(\vec{x}\vec{x}^T) \equiv Q_M(\vec{x})$, where $Q_M(\vec{x})$ denotes the quadratic form (as a function of $\vec{x} \in \mathbb{R}^d$) $Q_M(\vec{x}) \triangleq \vec{x}^T M\vec{x}$ and "$\equiv$" denotes the equality of two functions. Since, in the indexing phase of ONIAK, the input $\vec{y}$ (to $\eta(\cdot)$) is always a mapped data vector $f(\vec{x}) = \text{vec}(\vec{x}\vec{x}^T)$ for some $\vec{x} \in \mathcal{D}$, we have $\eta(\vec{y}) \equiv \vec{\theta}^T\text{vec}(\vec{x}\vec{x}^T) \equiv Q_\Theta(\vec{x})$, where the $d \times d$ kernel Θ is related to the d^2-dimensional vector $\vec{\theta}$ by $\vec{\theta} = \text{vec}(\Theta)$, and $\eta(\vec{y})$ is viewed as a function of $\vec{x}$. Now define $Z \triangleq (\Theta + \Theta^T)/2$. Since it is a well-known mathematical fact that $Q_Z(\vec{x}) \equiv Q_\Theta(\vec{x})$ (e.g., see [24] pp. 567), we have $\eta(\vec{y}) \equiv Q_Z(\vec{x})$.

Definition 2.3 ([5] pp. 6). A $d \times d$ random matrix Γ is GOE if

(1) Γ is symmetric and the scalars on the upper triangle are (mutually) independent RVs.
(2) Every scalar on its diagonal has distribution $\mathcal{N}(0, 1)$, and every scalar above its diagonal has distribution $\mathcal{N}(0, 1/2)$.

Since $\vec{\theta}$ is a d^2-dimensional Gaussian random vector (by the design of GP-LSH), Θ is a $d \times d$ *Gaussian random matrix*, defined as one whose d^2 scalars are i.i.d. standard Gaussian RVs. Hence, $Z = (\Theta + \Theta^T)/2$ is a GOE matrix defined above in Definition 2.3. Therefore, the (random function) $\eta(\vec{y})$ (viewed as a function of $\vec{x}$) in a GP-LSH function $h(\vec{y})$ can be replaced by a (random) GOE Quadratic form (GoeQF) $Q_Z(\vec{x})$ without changing any statistical property of $h(\vec{y})$. With this replacement, the problem of computing $\eta(\vec{y})$ becomes that of computing $Q_Z(\vec{x})$. In § 3, we will describe our FGoeQF technique that can compute a $Q_Z(\vec{x})$ (for any given argument $\vec{x}$), approximately but accurately as far as the efficacy of the GP-LSH function family is concerned in only $O(d \log d)$ time.

2.4 Low SNR Issue When d Is Large

In addition to the dimension blowup problem, we need to deal with another thorny issue when d is large. Recall that using this AMP ($f(\cdot)$ and $g(\cdot)$), we have reduced the ANNS-SALT problem to the ANNS-L_2 problem of searching for the nearest neighbor of $g(M)$ in $f(\mathcal{D})$. The standard solution approach to the latter problem is the aforementioned ONIAK, which is to index $f(\mathcal{D})$ using GP-LSH hash tables [12], against which $g(M)$ is searched with multi-probe enhancement [23]. The thorny issue is that when the dimension d is large (say $d > 200$), ONIAK may no longer outperform linear scan for the following reason: The signal-to-noise ratio (SNR) in this ANNS-L_2 problem is typically $O(1/d)$, which is very small when d is large. This $O(1/d)$ SNR issue is a fundamental limitation of the AMP design (that ONIAK unfortunately has to inherit) and was vaguely mentioned in [10]. For $d \leq 200$, ONIAK incorporates a shifting ($g(M)$ by a scaled identity matrix) heuristic proposed in [10] that can mitigate this issue to some extent, and the partitioning method [37] for getting the most out of the weak signal.

In this ANNS-L_2 problem, the $O(1/d)$ scaling of SNR is the result of AMP interacting with GP-LSH as follows. Let $\eta(\cdot)$ be a GP-LSH raw hash function as defined earlier. By (2) and the definition of $\eta(\cdot)$, $Y \triangleq \eta(f(\vec{x})) - \eta(g(M))$, the difference between the raw hash values of a data vector $f(\vec{x})$ and the query vector $g(M)$ in this ANNS-L_2 problem, is a zero-mean Gaussian RV with variance (energy) $\|\vec{x}\|_2^4 + 2\|M\vec{x}\|_2^2 + \|M^TM\|_F^2$. Suppose $\vec{x}^* \in \mathcal{D}$ minimizes $\|M\vec{x}\|_2$ or in other words $\vec{x}^*$ is the true nearest neighbor (of M) in "SALT distance". Then $Y^* \triangleq \eta(f(\vec{x}^*)) - \eta(g(M))$ is a zero-mean Gaussian RV with variance at least $\|\vec{x}\|_2^4 + \|M^TM\|_F^2$ (since $2\|M\vec{x}^*\|_2^2 \geq 0$ and $\|\vec{x}\|_2 = \|\vec{x}^*\|_2$ in a normalized dataset). Intuitively, the role $\eta(\cdot)$ plays in solving this ANNS-L_2 query is to tell (every other) $\vec{x}$ apart from $\vec{x}^*$ (so as to correctly output $\vec{x}^*$ as the final answer to the query), by telling apart two Gaussian RVs Y and Y^*. Telling apart these two Gaussian RVs is a textbook problem in signal processing:

Y^* ideally (with $\|M\vec{x}^*\| = 0$ being the ideal situation) contains only a Gaussian noise with energy $\|\vec{x}\|_2^4 + \|M^TM\|_F^2$ whereas Y is the sum of the Gaussian noise and a Gaussian signal with energy $2\|M\vec{x}\|_2^2 \geq 0$. Hence $2\|M\vec{x}\|_2^2/(\|\vec{x}\|_2^4 + \|M^TM\|_F^2)$ is roughly the SNR. It can be shown, under some mild assumptions (about the singular values of M), that $2\|M\vec{x}\|_2^2/(\|\vec{x}\|_2^4 + \|M^TM\|_F^2)$ (the SNR) scales as $O(1/d)$ when d grows larger.

As mentioned earlier, our solution to the case of large d (say $d > 200$) is JLT, whose outperformance over LS grows roughly as $O(d)$ (see Table 2). JLT does not outperform LS much (if any) when d is small (say $d < 64$). When d is moderately large (say between 64 and 200), we propose to combine ONIAK with JLT as follows. Given an (ANNS-ALT) query, we first obtain a list of candidates by querying the ONIAK hash tables (index). We then apply the JLT filter to these candidates, and compute the ground truth only for the candidates that survive the filtering.

2.5 Reduce ANNS-ALT to ANNS-SALT

Having described the ONIAK solution to ANNS-SALT, in this section we describe how an ANNS-ALT instance can be reduced to an ANNS-SALT instance in two steps: a *homogenization* step and a *normalization* step. Let $\mathcal{D}$ be the dataset in the ANNS-ALT instance and $\mathcal{D}''$ be the normalized homogenized dataset in the ANNS-SALT instance resulting from the reduction. $\mathcal{D}''$ is transformed from $\mathcal{D}$, using first a homogenization mapping ψ to be described in §2.5.1 and then a normalization mapping ϕ to be described in §2.5.2, as follows: Each vector $\vec{x}$ in $\mathcal{D}$ is mapped to a vector $\vec{x}'' = \phi(\psi(\vec{x}))$ in $\mathcal{D}''$, or in other words $\mathcal{D}'' \triangleq \phi(\psi(\mathcal{D}))$. The goal of the homogenization step (and the mapping $\psi(\cdot)$) is to ensure that, given any ANNS-ALT query $(M, \vec{q})$ over $\mathcal{D}$, we can correspondingly transform it to an ANNS-SALT query M'' over $\mathcal{D}''$ such that $M\vec{x} - \vec{q} = M''\vec{x}''$ (called *homogenized*) for any $\vec{x} \in \mathcal{D}$. As a result, for any $k > 0$, the top-k vectors in $\mathcal{D}$ that minimizes $\|M\vec{x} - \vec{q}\|_2$ (the true nearest neighbors for the ANNS-ALT query $(M, \vec{q})$) correspond precisely to the top-k vectors in $\mathcal{D}''$ that minimizes $\|M''\vec{x}''\|_2$ (the true nearest neighbors for the ANNS-SALT query M''). The goal of the normalization step (and and the mapping $\phi(\cdot)$) is to ensure that $\mathcal{D}''$ is normalized in the sense every mapped vector $\vec{x}'' = \phi(\psi(\vec{x}))$ in it has the same L_2 norm. Given a d-dimensional (which is the dimension of vectors in $\mathcal{D}$) ANNS-ALT problem, the ANNS-SALT problem thus reduced from it has dimension $d + 2$, since the homogenization step and the normalization step each expands the dimension of the problem by 1, as we will show. This tiny expansion clearly has negligible impact on query efficiency and accuracy.

2.5.1 Homogenization. In this step, we map $\mathcal{D}$ to a homogenized (but not yet normalized) dataset $\mathcal{D}'$ using the homogenization mapping $\psi(\cdot)$ as follows. For each $\vec{x}$ in $\mathcal{D}$, we obtain the corresponding $\vec{x}' = \psi(\vec{x})$ (in $\mathcal{D}'$) by appending a scalar of value 1 to the bottom of (the column vector) $\vec{x}$; in other words we define $(\vec{x}')^T \triangleq [\vec{x}^T; 1]$. Given each ANNS-ALT query $(M, \vec{q})$, we transform it to a homogenized query M', using a companion mapping of $\psi(\cdot)$ as follows. We obtain M' by appending $-\vec{q}$ as a new column to the right of M; or formally $M' \triangleq [M; -\vec{q}]$. It is not hard to verify that the following homogenization goal is achieved: $M\vec{x} - \vec{q} = M'\vec{x}'$. In computer graphics, the technique of mapping $(M, \vec{q})$ to M' is called *homogeneous coordinates* [10, 30].

2.5.2 Normalization. In the normalization step, we map $\mathcal{D}'$, the homogenized dataset, to the aforementioned $\mathcal{D}''$ that is both homogenized and normalized, using the aforementioned normalization mapping $\phi(\cdot)$. Let V be an upper bound on the L_2 norms of all data items in $\mathcal{D}'$. For each $\vec{x}'$ in $\mathcal{D}'$, we obtain the corresponding $\vec{x}'' = \phi(\vec{x}')$ (in $\mathcal{D}''$) by appending a scalar of value $\sqrt{V^2 - \|\vec{x}'\|_2^2}$ to the bottom of (the column vector) $\vec{x}'$; in other words, we define $(\vec{x}'')^T \triangleq \left[(\vec{x}')^T; \sqrt{V^2 - \|\vec{x}'\|_2^2}\right]$. It is not hard to verify that every $\vec{x}''$ (thus normalized) in $\mathcal{D}''$ has the same L_2-norm of value V. Hence $\mathcal{D}''$ is normalized. Since each $\vec{x}'$ has been changed to $\vec{x}''$ during this normalization step, we need to correspondingly map (using a companion mapping of $\phi(\cdot)$) a homogenized query M' to a query M'' that remains homogenized in the sense $M''\vec{x}'' = M'\vec{x}' = M\vec{x} - \vec{q}$. To this end, we obtain M'' by appending a column of zeros to the right of M'; in other words, we define $M'' \triangleq [M'; \vec{0}]$. It is not hard to verify that M'' remains homogenized in the above sense.

2.6 Scale ONIAK to Massive Datasets

In this subsection, we discuss how ONIAK can scale to massive datasets that are too large to fit in memory and have to reside on fast external memory such as solid-state drive (SSD) or non-volatile memory (NVM) [26]. As explained earlier, ONIAK uses GP-LSH hash tables with multi-probe [23], the design objective of which is to minimize the index size. Hence its index can typically fit in main memory (even when the dataset cannot), from which the list of (ANNS-SALT) candidates can be obtained at high speed. Although the remaining step of computing the ground truth has to read these candidates from the fast external memory, the performance bottleneck of this step is typically the $O(d^2)$ per-candidate time complexity of the ground truth computation. In this case, the query efficiency will not degrade much. Under rare circumstances in which a dataset is so large that even this index cannot fit in memory, the resulting (from the AMP) ANNS-L_2 problem can instead be solved using external memory LSH solutions, such as SRS [31], QALSH [15] and PM-LSH [40]. In this case, FGoeQF plays an important role in reducing the indexing time since all these solutions use GP-LSH functions.

3 FAST GOE QUADRATIC FORM (FGOEQF)

For ease of presentation, in this section we assume the dimension d is a power of 2; if this is not the case, we can make it be at the cost of inflating d by at most a factor of 2. Our FGoeQF technique, which can evaluate a GoeQF approximately in only $O(d \log d)$ time, works as follows. We can generate and fix (an instance of) a special $d \times d$ random matrix Z' that satisfies two nice properties: (1) Z' is approximately distributed as GOE (see Definition 2.3); (2) evaluating each $Q_{Z'}(\cdot)$ has a time complexity of only $O(d \log d)$. Our FGoeQF technique is simply to use such a $Q_{Z'}(\cdot)$ instance instead whenever a precise GoeQF instance is needed.

This "almost GOE" matrix Z' is defined as $Z' \triangleq (R')^T \Lambda R'$, where R' and Λ are independent $d \times d$ random matrices, Λ is a diagonal matrix drawn from the eigenvalue distribution of the GOE (see Definition 3.1), and R' is a uniformly pseudorandom rotation proposed in [6] as a cheaper replacement for a (truly) *uniformly random rotation* R (see Definition 3.2). This R' is defined as $R' \triangleq HD_3HD_2HD_1$,

where H is the $d \times d$ (normalized) Hadamard matrix H_d defined below in Definition 3.3. Here D_1, D_2, and D_3 are three independent random diagonal matrices, and in each D_i, $i = 1, 2, 3$, the scalars on the diagonal are i.i.d. RVs that each takes values -1 and 1 each with probability $1/2$. R' is an orthonormal matrix (see Definition 3.2, a.k.a. roation) because H and each D_i is, and the product of orthonormal matrices is also orthonormal.

Definition 3.1 ([24] pp. 517). The *spectral decomposition* of a symmetric matrix M takes the form $R^T \Lambda R = M$, where R is an orthonormal matrix and Λ is a diagonal matrix. If M is a random matrix (say, with distribution $\mathcal{M}$), then both Λ and R are also random matrices. We call the distribution of Λ the *eigenvalue distribution* of $\mathcal{M}$.

Definition 3.2. A $d \times d$ matrix M is said to be *orthonormal* if $M^T M = I_d$. Every $d \times d$ orthonormal matrix corresponds to a *rotation* on the Euclidean space $\mathbb{R}^d$, and vice versa. The set of $d \times d$ orthonormal matrices form a topological group called the *orthogonal group* $OG(d)$, on which probability measures can be rigorously defined. The uniform distribution over $OG(d)$, called the *Haar measure*, is defined as the unique measure that is invariant under rotations (see Theorem F.13 in [5]). In other words, a *uniformly random rotation* is defined as an orthonormal matrix that is distributed by the Haar measure on $OG(d)$.

Definition 3.3. The normalized *Hadamard matrix* is defined as follows. For the base case, the 2×2 Hadamard matrix H_2 is defined as $1/\sqrt{2} \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. Then, for any d that is a power of 2, the $d \times d$ Hadamard matrix H_d is defined recursively as $1/\sqrt{2} \cdot \begin{bmatrix} H_{d/2} & H_{d/2} \\ H_{d/2} & -H_{d/2} \end{bmatrix}$.

Time complexity analysis. Given any input vector $\vec{x}$, the quadratic form $Q_{Z'}(\vec{x}) = \vec{x}^T R'^T \Lambda R' \vec{x} = (R'\vec{x})^T \Lambda (R'\vec{x}) = \vec{u}^T \Lambda \vec{u}$ can be computed in $O(d \log d)$ time as follows. Here $\vec{u} \triangleq R'\vec{x} = HD_3HD_2HD_1\vec{x}$. We can perform the six matrix-vector multiplications in $HD_3HD_2HD_1\vec{x}$ from right to left (to obtain the value of $\vec{u}$) in $O(d \log d)$ time since multiplying each D_i, $i = 1, 2, 3$, to a vector takes $O(d)$ time, and multiplying H to a vector takes $O(d \log d)$ time, using the fast Walsh-Hadamard transform (FWHT), which is a special form of fast Fourier transform (FFT) [1, 2]. Once we have the value of $\vec{u}$, we can compute $Q_{Z'}(\vec{x}) = \vec{u}^T \Lambda \vec{u} = \sum_{i=1}^{d} (\lambda_{i,i}) u_i^2$ in $O(d)$ time.

Theoretical justification. The random matrix Z' used in FGoeQF is close to a GOE matrix in distribution for the following reason. Let Z be a $d \times d$ matrix defined as $Z \triangleq R^T \Lambda R$, where R is a uniformly random rotation, and Λ is a diagonal matrix drawn from the eigenvalue distribution of the GOE. Theorem 3.4 below implies that Z is distributed exactly as a GOE matrix. Comparing the random matrix $Z' \triangleq R'^T \Lambda R'$ with Z, it is clear that their only difference is that a uniformly pseudorandom rotation $R' \triangleq HD_3HD_2HD_1$ is used in defining Z' whereas a (truly) uniformly random rotation (see Definition 3.2) R is used in defining Z. However, it was shown empirically in [6] and theoretically in [11] that R' is very close to R in distribution. Hence, Z' is very close to Z (GOE) in distribution.

Theorem 3.4 (Corollary 2.5.4 in [5]). *Let M be a GOE random matrix, and $M = R_M^T \Lambda_M R_M$ be its spectral decomposition. Then the following properties hold:*

1. *Λ_M and R_M are independent random matrices.*
2. *R_M is a uniformly random rotation, or in other words a random orthonormal matrix distributed by the Haar measure.*

Empirical verification. It was shown in [12] that all statistical properties of, and consequently all the accuracy and the efficiency guarantees of the GP-LSH scheme, can be derived from the following distributional property of a GP-LSH raw hash function $\eta(\vec{y}) = \vec{\theta}^T \vec{y}$: Given any input vector $\vec{y}$, the RV $\vec{\theta}^T \vec{y}$ (whose randomness comes from the randomness of $\vec{\theta}$) needs to be distributed as $\mathcal{N}(0, \|\vec{y}\|_2^2)$. Translated into the ONIAK context, where $\vec{y} = \text{vec}(\vec{x}\vec{x}^T)$ and hence $\|\vec{y}\|_2^2 = \|\vec{x}\|_2^4$, replacing $\eta(\vec{y})$ with $Q_{Z'}(\vec{x})$ is harmless to GP-LSH as long as the distribution of $Q_{Z'}(\vec{x})$ is close to $\mathcal{N}(0, \|\vec{x}\|_2^4)$. We have verified this closeness through extensive Monte Carlo simulations.

FJLT – An alternative to FGoeQF. As mentioned in §2.3, $\eta(\vec{y}) \equiv \eta(\text{vec}(\vec{x}\vec{x}^T)) \equiv Q_\Theta(\vec{x})$, where Θ is a Gaussian random matrix. We have discovered another technique for computing $Q_\Theta(\vec{x})$ approximately also in $O(d \log d)$ time. This technique is based on a theory result, called fast Johnson-Lindenstrauss transform (FJLT) [2], for generating a pseudorandom matrix Θ' that has two properties: (i) Θ' is close to Θ in distribution; and (ii) given any input $\vec{x}$, $\Theta'\vec{x}$ and hence $Q_{\Theta'}(\vec{x}) = \vec{x}^T \Theta' \vec{x}$ can be computed in $O(d \log d)$ time. Our FJLT-based technique is simply to generate and use such a $Q_{\Theta'}(\cdot)$ wherever a $Q_\Theta(\cdot)$ is called for. However, this FJLT-based technique is not our preferred scheme, because it is at least an order of magnitude slower than FGoeQF when d is large (say $d \geq 1000$), according to our measurements.

4 BABY PROBLEMS OF ANNS-ALT

In this section, we describe all baby problems of ANNS-ALT that we are aware of. They include the aforementioned ANNS in Mahalanobis distance (§4.1), ANNS in weighted L_2 distance (§4.2), and subspace-to-point ANNS problems (§4.3). Each subspace-to-point ANNS problem has a dual form. As readers might have guessed, each such dual form can be reduced to, and solved as, a dual problem of ANNS-ALT as we will show in §4.4.

4.1 ANNS in Mahalanobis Distance (ANNS-M)

Both static (with a fixed kernel) and general (with changing kernels) ANNS-M have a wide range of applications in data analytics such as k-nearest neighbor (kNN) classification [14, 20, 39], outlier detection [28] and clustering [34]. Since we have described static ANNS-M in § 1.1, here we focus on general ANNS-M and describe two application scenario types of it: (1) online learning; and (2) many different learning jobs over the same (massive) dataset running concurrently.

In an online Mahalanobis learning scenario, the arrival of training data is a gradual and continuous process, and so is the process of learning the right kernel. Initially, the starting kernel Σ_0 is the identity matrix (in which case the Mahalanobis distance is the same as the L_2 distance). Then, after a set of new training instances (data items and their labels) $\mathcal{T}_1$ arrive, the learning process learns a more accurate kernel Σ_1 incrementally, by modifying Σ_0 [39] to fit $\mathcal{T}_1$. The fitting procedure involves performing ANNS-M queries in which

the query kernel is Σ_0 and the query vectors are $\mathcal{T}_1$. In this incremental way, we learn, as training instances stream in, a sequence of kernels Σ_i, $i = 1, 2, \cdots$, by processing a corresponding sequence of dynamic ANNS-M queries with kernels Σ_{i-1}, $i = 1, 2, \cdots$. Σ_i usually gradually converges to the right kernel for the application when i becomes larger.

If we were to use the aforementioned (naive) Rebuild LSH approach for processing such a sequence of ANNS-M queries, then when the kernel changes, say, from $\Sigma_i = U_i^T U_i$ to $\Sigma_{i+1} = U_{i+1}^T U_{i+1}$, the current index on $U_i(\mathcal{D})$ needs to be scratched, and a new index on $U_{i+1}(\mathcal{D})$ needs to be built, which as explained earlier is an expensive process. Intuitively there is considerable wasted effort here, since in an online Mahalanobis learning application, the difference between Σ_i and Σ_{i+1} can be quite small [39]. OASIS [39] minimizes such waste by taking full advantage of this "locality pattern" as follows: Whenever the difference between Σ_i and Σ_{i+1} is small, the current index on $U_i(\mathcal{D})$ can continue to be used lazily for processing ANNS-M queries with the new kernel Σ_{i+1}, at a small accuracy loss, so that the expensive re-indexing process can be avoided.

The other application scenario type is to concurrently run many different learning jobs over the same dataset $\mathcal{D}$ that each generates a sequence of static ANNS-M queries over $\mathcal{D}$. The kernels used by these jobs can be very different from one another, since these jobs can have completely different objectives and depend on very different subsets of coordinates. Since these jobs are concurrent, their respective sequences of static ANNS-M queries interleave with one another to form an aggregate sequence of general ANNS-M queries. To process this aggregate query sequence, we can certainly treat each such static ANNS-M query sequence as an ANNS-L_2 query sequence as explained in §1.1. However, to do so, for processing each such sequence (say with kernel $\Sigma = U^T U$) we need to store (in memory) a separate index (comprised of the mapped dataset $U(\mathcal{D})$ and GP-LSH hash tables for indexing $U(\mathcal{D})$) that is clearly larger than the size of $\mathcal{D}$. When the number of jobs and the size of $\mathcal{D}$ are both large, the total size of these indices can become too large to fit in main memory.

In comparison, when d is not large, ONIAK is a more scalable solution approach, since we can process the entire aggregate query sequence using a single universal index that is typically *much smaller* (explained in §2.6) than the size of $\mathcal{D}$; and when d is large, JLT is a more scalable approach since it does not require an index.

We now describe an example application scenario of this type, the objective of which is to learn the relationships between *gene expression level* and various diseases. Human gene expression level (GEL) is a vector of about 20,000 dimensions, each scalar of which corresponds to one gene [3]. Each GEL instance corresponds to cells sampled from a human subject, in which each scalar is a quantitative measure of how active the corresponding gene is in the cells. A fundamental problem in life sciences is to find the causal relationship between a GEL instance and the behavior of the corresponding cells (called *phenotype*) with respect to a disease of interest. Now imagine that a medical research institute has collected a dataset $\mathcal{D}$ containing a large number of GEL instances (vectors). $\mathcal{D}$ is queried by many medical research teams around the world concurrently. Each research team studies a certain disease, and its learning job generates a stream of static ANNS-M queries over $\mathcal{D}$ concerning the disease. Since each job concerns a different disease, the Mahalanobis kernels used in these jobs are generally different from one another.

4.2 ANNS in Weighted L_2 Distance

Another baby problem of ANNS-ALT is ANNS in weighted L_2 distance, or ANNS-WL_2 for short. A weighted L_2 distance metric (for $\mathbb{R}^d$) is parameterized by a weight vector $\vec{\lambda}$. For any two d-dimensional vectors $\vec{x}$ and $\vec{y}$, the weighted L_2 distance between them is defined as $\left(\sum_{i=1}^{d} \lambda_i^2 (x_i - y_i)^2\right)^{1/2}$. In ANNS-W$L_2$, the weight vectors are generally different across different queries, which makes it a much harder problem than (unweighted) ANNS-L_2. ANNS-WL_2 is a special case of dynamic ANNS-M in which each kernel is a (different) diagonal matrix.

Before this work, SL-ALSH [19] is the only solution to ANNS-WL_2 in the literature. Like ONIAK, SL-ALSH also first employs an AMP to reduce the original ANNS problem (in this case ANNS-WL_2) to ANNS-L_2, and then builds an index for the resulting ANNS-L_2. However, the AMP used in SL-ALSH is fundamentally different than that used in ONIAK in the following way: Whereas the AMP in ONIAK (see §2.1) is strictly order-preserving between any aforementioned trio of vectors $\vec{x}_1$, $\vec{x}_2$, and $\vec{q}$, as we have proved in (2), AMP in SL-ALSH is approximately order-preserving only when every scalar in this trio has a small absolute value. This is because SL-ALSH approximates the function $\cos(t)$ using $1 - t^2/2$, the first two terms of its Taylor series, and the approximation error is small only when t is close to 0. We will compare the efficacies of ONIAK and SL-ALSH in processing ANNS-WL_2 queries in §6.2.

4.3 Subspace-to-Point Problems

In this section, we describe three closely related subspace-to-point (S2P) ANNS problems that are also baby problems of ANNS-ALT. For ease of presentation, we adopt the convention that every point or subspace lies in a d-dimensional Euclidean space (d is called the *ambient dimension*). In S2P ANNS, the dataset $\mathcal{D}$ contains a large number of d-dimensional vectors like before, and the query S is a ρ-dimensional ($1 \le \rho < d$) affine subspace of $\mathbb{R}^d$. The goals of these three S2P ANNS problems are to find $\vec{x} \in \mathcal{D}$ such that (1) $dist(\vec{x}, S)$ (the L_2 distance from the query subspace S to a point $\vec{x}$) is the smallest (called *MinDistS2P*); (2) the projection of $\vec{x}$ onto S, denoted as $P_S(\vec{x})$, has the smallest L_2 norm (called *MinProjS2P*); and (3) $P_S(\vec{x})$ has the largest L_2 norm (called *MaxProjS2P*), respectively. Among the three, MinDistS2P has many applications in computer vision [10, 32]. We note that AMP-based solutions for MinDistS2P were proposed in [10], but no efficient solution exists for MinProjS2P and MaxProjS2P until this work.

In the following, we show how each of these three S2P ANNS problems can be reduced to ANNS-ALT. To this end, we first state an elementary fact in linear algebra: Every ρ-dimensional ($1 \le \rho < d$) affine subspace S can be represented as a set of d-dimensional vectors in the form of $\{\vec{z} \mid A\vec{z} = \vec{b}\}$, where $\vec{b}$ is a $(d - \rho)$-dimensional vector, and A is a $(d - \rho) \times d$ matrix with orthonormal rows in the sense that AA^T is equal to the $(d - \rho) \times (d - \rho)$ identity matrix $I_{d-\rho}$ [24]. Every affine subspace that appears in the sequel is assumed to have such an orthonormalized representation.

4.3.1 Reduce MinDistS2P to ANNS-ALT. By Theorem 4.1, a MinDistS2P query, in which the query subspace S is the same as the S defined in Theorem 4.1, is equivalent to the following ANNS-ALT query: The query matrix is $M = A^T A$ and the query vector is $\vec{q} = A^T\vec{b}$; the objective is to find $\vec{x} \in \mathcal{D}$ such that $\|M\vec{x} - \vec{q}\|_2$ is minimized. We note the AH2PS problem [32] described in §2.1 is a special case of MinDistS2P, in which each query subspace is a $(d-1)$-dimensional linear subspace (aka. *hyperplane*) in the form $\{\vec{z} \mid \vec{a}^T\vec{z} = 0\}$.

Theorem 4.1 ([24] pp. 430). *Let $S = \{\vec{z} \mid A\vec{z} = \vec{b}\}$ be an affine subspace of $\mathbb{R}^d$ and $\vec{x}$ be a d-dimensional vector. Then $dist(\vec{x}, S)$ (in L_2) is equal to $\|A^T A\vec{x} - A^T\vec{b}\|_2$.*

4.3.2 Reduce MinProjS2P to ANNS-ALT. By Theorem 4.2(1), a MinProjS2P query, in which the query subspace S is the same as the S defined in Theorem 4.2, is equivalent to the following ANNS-ALT query: The query matrix is $M = I_d - A^T A$ and the query vector is $\vec{q} = \vec{0}$; the objective is to find $\vec{x} \in \mathcal{D}$ such that $\|M\vec{x}\|_2$ is minimized.

Theorem 4.2 ([24] pp. 430). *Let $S = \{\vec{z} \mid A\vec{z} = \vec{b}\}$ be any affine subspace of $\mathbb{R}^d$ and $\vec{x}$ be any d-dimensional vector. We have*

1. *$P_S(\vec{x}) = (I_d - A^T A)\vec{x}$.*
2. *If S is a linear subspace (defined as an affine subspace that contains the $\vec{0}$ vector), then we have $\|\vec{x}\|_2^2 = \|P_S(\vec{x})\|_2^2 + dist^2(\vec{x}, S)$ by the Pythagorean theorem.*

4.3.3 Reduce MaxProjS2P to ANNS-ALT. Here we only show how to reduce MaxProjS2P to MinDistS2P, which can be reduced to ANNS-ALT as explained earlier. We first state a simple mathematical fact needed in this reduction: Let $S = \{\vec{z} \mid A\vec{z} = \vec{b}\}$ be any affine subspace of $\mathbb{R}^d$, and $S' = \{\vec{z} \mid A\vec{z} = \vec{0}\}$ be the linear subspace parallel to S; then given any d-dimensional vector $\vec{x}$, we have $P_S(\vec{x}) = P_{S'}(\vec{x})$. Thanks to this fact, we can assume that the query S is a linear subspace. We let $S = \{\vec{z} \mid A\vec{z} = \vec{0}\}$ in the sequel.

For the MaxProjS2P-to-MinDistS2P reduction, we need the dataset $\mathcal{D}$ to be normalized (every $\vec{x} \in \mathcal{D}$ has the same L_2-norm). Then, MaxProjS2P is equivalent to MinDistS2P, since the $\vec{x} \in \mathcal{D}$ that maximizes $\|P_S(\vec{x})\|_2$ is the same as that minimizes $dist(\vec{x}, S)$, due to Theorem 4.2(2).

4.4 Dual Point-to-Subspace Problems

All three (primal) subspace-to-point ANNS problems have corresponding dual problems, in which the roles of the data items (in the dataset $\mathcal{D}$) and the query are reversed. We refer to these dual problems as point-to-subspace (P2S) ANNS. In P2S ANNS, the dataset $\mathcal{D}^*$ contains a large number of affine subspaces $S_1, S_2, \cdots, S_n \subseteq \mathbb{R}^d$, whereas each query is a d-dimensional vector $\vec{x}$. The three P2S ANNS problems are to find the affine subspace $S \in \mathcal{D}^*$ such that (1) $dist(\vec{x}, S)$ is the smallest (this problem is the dual of MinDistS2P, and we call it *MinDistP2S*); (2) the projection $P_S(\vec{x})$) has the smallest L_2 norm (this problem is the dual of MinProjS2P, and we call it *MinProjP2S*); and (3) $P_S(\vec{x})$ has the largest L_2 norm (this problem is the dual of MaxProjS2P, and we call it *MaxProjP2S*), respectively. Like for the primal cases, AMP-based solutions for MinDistP2S were proposed in [10], but no efficient solution exists for MinProjP2S and MaxProjP2S until this work.

All three dual problems are baby problems of the dual of the ANNS-ALT that we call ANNS-ALTD, in which the formats of the data items and the query are similarly reversed. In ANNS-ALTD, the dataset $\mathcal{D}^*$ is a set of tuples in the form of $\{(M_1, \vec{q}_1), (M_2, \vec{q}_2), \cdots, (M_n, \vec{q}_n)\}$ whereas each query is a vector $\vec{x}$. The objective of ANNS-ALTD is to find $(M, \vec{q}) \in \mathcal{D}^*$ that minimizes $\|M\vec{x} - \vec{q}\|_2$. ANNS-ALTD can be solved in a way that can be considered the dual of ONIAK, as we will describe shortly.

In the interest of space, we show only how one of the P2S ANNS problems, namely MinDistP2S, is reduced to ANNS-ALTD. Like in the formulation of the primal problems, we assume every affine subspace in the MinDistP2S dataset $S \in \mathcal{D}_1^*$ is in the form $\{\vec{z} \mid A\vec{z} = \vec{b}\}$, and the query is $\vec{x}$. Since $dist(\vec{x}, S) = \|A^T A\vec{x} - A^T\vec{b}\|_2$ (shown in Theorem 4.1), the MinDistP2S problem can be viewed as the following ANNS-ALTD instance: The ANNS-ALTD dataset $\mathcal{D}_2^*$ (transformed from $\mathcal{D}_1^*$) is $\{(M_1, \vec{q}_1), (M_2, \vec{q}_2), \cdots, (M_n, \vec{q}_n)\}$, in which $M_j = A_j^T A_j$ and $\vec{q}_j = A_j^T\vec{b}_j$, for $j = 1, 2, \cdots, n$; and the query is also $\vec{x}$.

Finally, we describe our "ONIAK-dual" and "JLT-dual" solutions to ANNS-ALTD. Like how we reduce ANNS-ALT to ANNS-SALT, we can similarly reduce ANNS-ALTD to a simple form that we refer to as simple ANNS-ALTD, or ANNS-SALTD in short; we omit the detail of the reduction in the interest of space. In ANNS-SALTD, the dataset $\mathcal{D}^*$ is simply a set of matrices $\{M_1, M_2, \cdots, M_n\}$, and the objective is to find $M \in \mathcal{D}^*$ that minimizes $\|M\vec{x}\|_2$ given a query $\vec{x}$; in addition, each matrix M in $\mathcal{D}^*$ is assumed to be normalized in the sense $\|M^T M\|_F$ is the same for every $M \in \mathcal{D}^*$. Our ONIAK-dual solution to this ANNS-SALTD problem (when d is not too large) works as follows. Recall that in the (primal) ONIAK solution to (primal) ANNS-SALT, we index $f(\mathcal{D}) \triangleq \{\text{vec}(\vec{x}\vec{x}^T) \mid \vec{x} \in \mathcal{D}\}$ using GP-LSH, and search $g(M) = -\text{vec}(M^T M)$ (given a query M) against the GP-LSH index. In ONIAK-dual, we instead index $g(\mathcal{D}^*) \triangleq \{-\text{vec}(M^T M) \mid M \in \mathcal{D}^*\}$ using GP-LSH, and search $f(\vec{x}) = \text{vec}(\vec{x}\vec{x}^T)$ (given a query $\vec{x}$) against the GP-LSH index. Our JLT-dual solution (when d is large) is simply to "JLT" each M_j to $M_j' = \Theta M_j$ (that contains only $l \ll d$ rows) at the indexing stage and store these $(M_j')'s$ as its index. Then, like in our JLT solution, we perform relatively cheap $M_j'\vec{x}$ computations to filter out most of unpromising candidates.

5 ANNS-M PERFORMANCE EVALUATION

Recall that our ANNS-ALT solution is ONIAK when $d \leq 200$ and is JLT (linear scan with JLT filtering) otherwise. In this section, we evaluate the efficacies of ONIAK and JLT for ANNS-M only. We compare them against OASIS [39], the state-of-the-art ANNS-M solution, and linear scan (LS). As will be detailed in §5.3, we use ANNS-M workloads in which (Mahalanobis) kernels used in successive ANNS-M queries often differ very little from one another. Such workloads put OASIS at an advantage in its performance comparison with all other algorithms, since only OASIS can take advantage of such "locality patterns".

5.1 Experiment Settings

5.1.1 Evaluation Datasets and Workload. We use a similar set of datasets as those used in evaluating OASIS [39]. These datasets are

listed in Table 1. They include a low-dimensional real-world dataset named HEPMASS-27 [8] and three high-dimensional synthetic datasets named SYN-254, SYN-510 and SYN-1024, respectively. All datasets are labeled for binary classification, and their labels are used to generate workloads (Mahalanobis query matrices) based on an online (Mahalanobis) learning scenario (described in §4.1) using the same code and under the same settings as in the evaluation of OASIS in [39].

We use three synthetic datasets because of the difficulty in finding large high-dimensional datasets that have binary labels [39]. These datasets (both data and query points and their labels) are generated using the `make_classification` method in scikit-learn [29] as suggested by the authors of OASIS [39]. In this way, the online Mahalanobis learning (used in workload generation) is a nontrivial process and as a result, the $n_q = 100$ ANNS-M queries have different (gradually changing) Mahalanobis kernels.

Table 1: Datasets used in ANNS-M evaluation.

Dataset	n	d	n_q	Type
HEPMASS-27 [8]	10.0M	27	100	Physics
SYN-254	1.0M	254	100	Synthetic
SYN-510	1.0M	510	100	Synthetic
SYN-1022	1.0M	1,022	100	Synthetic

5.1.2 Performance Metrics. We mainly evaluate the query efficiency at a similar query accuracy. Query efficiency is measured by the average *gross query time* over the 100 queries. Recall that OASIS needs to perform re-indexing (re-projection plus re-hashing) when the Mahalanobis kernel Σ_i differs significantly from the last kernel Σ_{i-1} as explained in §4.1. We divide the gross query time of OASIS into two parts: the (total) *re-indexing time* and the (total) *net query time*. For ONIAK, LS, and JLT, the *gross query time* is equal to the *net query time* since they have no re-indexing operations.

We measure query accuracy mainly by recall@50, which is defined as follows: For a query tuple $(\Sigma, \vec{q})$, let the query result of the first $k = 50$ nearest neighbors (of $\vec{q}$) in the Mahalanobis distance D_M^{Σ} be a set R, and R^* be the set of actual k nearest neighbors to $\vec{q}$ in D_M^{Σ}. Then, recall@50 is equal to $|R \cap R^*|/k$. We also measured recalls under any other values of k ranging from 1 to 100 (commonly used in the ANNS literature [21, 39]). The results are similar for most of our experiments.

5.1.3 JLT Parameter Settings. We use JLT on the three high-dimensional datasets SYN-254, SYN-510 and SYN-1022. Recall that JLT maps a $d \times d$ query matrix M to an $l \times d$ matrix using an $l \times d$ Gaussian random matrix Θ (described in §1.3). In our experiments, we set $l = 15$ when $d = 254$, $l = 30$ when $d = 510$, and $l = 40$ when $d = 1022$.

5.2 Implementation Details

5.2.1 Re-implementation of Parts of OASIS in C++. Since OASIS [39] is implemented in Python, for a fair comparison of running times, we re-implement in C++ all code components of OASIS for processing an ANNS-M query except the following three: (i) the re-hashing operation, as a part of the re-indexing operation; (ii) obtaining the list of ANNS-M candidates (by hashing the query vector and reading out the contents of the corresponding hash buckets), as a part of ANNS-M query processing; and (iii) the computation of ground truth (ANNS-M distances between the query point and the ANNS-M candidates), also as a part of ANNS-M query processing. We now explain how we handle these three components. Component (i) includes two parts: computing the GP-LSH hash values (for each $\vec{x} \in \mathcal{D}$) and inserting each such $\vec{x}$ into the GP-LSH hash tables. We reimplement in C++ the first part and measure its running time. We do not "charge" OASIS for the second part to OASIS' advantage. We also do not "charge" OASIS for component (ii). As to component (iii), its running time can be accurately estimated as follows. We have already implemented using C++ the code (common for OASIS, ONIAK, JLT, and LS) for computing a single ground truth distance, which involves a matrix-vector multiplication, and accurately measured its running time that we denote as δ. To arrive at the running time of component (iii) for processing an ANNS-M query, we first obtain the *exact* number of candidates from its Python run and multiply this number to δ. This estimation is accurate, since each such matrix-vector multiplication involves the same number of arithmetic operations. To summarize, we "charge OASIS strictly in C++" if we "charge" OASIS at all.

We now state two facts that we will use in §5.3. The first fact is that a linear scan and a re-projection operation take exactly the same amount of time $n\delta$ to perform since both involve the same n matrix-vector multiplications, where $n = |\mathcal{D}|$. The second fact is that the running time of OASIS component (iii), which we deem the *net query time* (defined next) of OASIS in its performance reporting (to OASIS' advantage), is equal to that of a linear scan ($n\delta$) multiplied by the selectivity that we denote as γ, where selectivity is defined in the ANNS literature (e.g., [27]) as the percentage of data vectors in $\mathcal{D}$ that are selected to be ANNS candidates (of which the ground truth is computed for reaching a target recall).

5.2.2 Other Implementation Details. We implement ONIAK in C++ based on the open-source `FALCONN` [25] library. Both the matrix-matrix (for JLT) and the matrix-vector (for ground truth) multiplications are implemented using the highly optimized C++ library `Eigen` [17]. We compile all C++ source code using g++ 7.5 with -O3. In this work, all experiments are conducted on a workstation running Ubuntu 18.04 with Intel(R) Core(TM) i7-9800X 3.8 GHz CPU, 128 GB DRAM and 4 TB hard disk drive (HDD).

Table 2: Query time (in seconds) with recall@50 > 0.8.

Dataset	ONIAK	OASIS	OASIS+JLT	LS	JLT
HEPMASS-27	0.85	0.34+0.07	-	1.37	-
SYN-254	-	2.52+0.64	2.52+0.07	5.87	0.54
SYN-510	-	10.52+3.01	10.52+0.18	23.13	1.16
SYN-1022	-	71.17+26.56	71.17+0.62	166.02	3.31

5.3 Experimental Results

Table 2 shows the average query times of all benchmark algorithms. For a more insightful comparison, for OASIS, we break down each

(gross) query time into two parts: the re-indexing time (t_1) and the net query time (t_2). The *net query time* of OASIS should include the running times of the aforementioned code components (ii) and (iii), and the amount of time it takes to hash the query vector. However, in Table 2 we "charge" OASIS only for (iii) to OASIS' advantage. Each query time of OASIS in Table 2 is written as "$t_1 + t_2$". For all solutions, each query time in Table 2 is the average over 100 corresponding ANNS-M queries.

As explained earlier, our solution is ONIAK for HEPMASS-27, since JLT does not offer any performance boost for a small d of 27, and our solution is JLT for SYN-254, SYN-510, and SYN-1022, since their dimensions are all larger than 200. We did not measure the query times of other combinations, so the corresponding entries in Table 2 are marked as "-". As shown in Table 2, OASIS has a shorter query time than ONIAK on HEPMASS-27. This outperformance of OASIS is due to two factors. The first factor is that the "custom-built" (for a kernel Σ_i) index of OASIS is better than the universal index (for all kernels) of ONIAK: On HEPMASS-27, the average selectivity (defined in § 5.2.1) for OASIS is very small at 0.05 (= 0.07/1.37 as explained in § 5.2.1), and that for ONIAK is quite large at 0.48. On the three higher-dimensional datasets, OASIS' selectivities are slightly higher at 0.11, 0.13, and 0.16 respectively, which is consistent with the fact that the ANNS-L_2 query (over $U_i(\mathcal{D})$) becomes increasingly more difficult to process when d becomes larger. The second factor is that over the HEPMASS ANNS-M workload, OASIS has a very low "miss ratio" of 0.1 (i.e., 10 re-indexing operations over $n_q = 100$ queries) so that its average re-indexing cost is kept low at 0.34s. Had this "miss ratio" increased to 0.23, OASIS would have underperformed ONIAK.

As shown in Table 2, our solution JLT has much shorter query times than OASIS on the three higher-dimensional datasets and this outperformance grows roughly linear with d: 5.8, 11.7, and 29.5 times shorter on SYN-254, SYN-510, and SYN-1022 respectively. On these three datasets, OASIS' "miss ratios" are higher at 0.32, 0.37, and 0.4 respectively, but JLT's outperformance does not depend on that. For example, JLT would outperform OASIS by 2.7, 5.0, and 13.4 times respectively even if all three "miss ratios" were reduced to 0.1 (as in the case of HEPMASS-27). In fact, JLT outperforms OASIS by 1.2, 2.6, 8.0 times respectively even in the ideal (to OASIS) case of zero "miss ratio".

We discover that JLT can be used to reduce the net query time t_2 of OASIS, by JLT-filtering the ANNS-M candidates "nominated" by OASIS. As shown in the "OASIS+JLT" column in Table 2, JLT reduces t_2 by 1.2, 1.3, and 1.4 times respectively on SYN-254, SYN-510, and SYN-1022. OASIS+JLT cannot be applied to HEPMASS-27 since JLT does not help when $d = 27$.

6 OTHER EVALUATIONS

In this section, we evaluate the efficacy of ONIAK for two other baby problems, namely MinDistS2P (described in § 4.3) and ANNS-WL_2 (described in § 4.2). We keep both evaluations separate from the ANNS-M evaluation, since the ANNS-M evaluation uses very different workloads that have "locality patterns" irrelevant to both evaluations. For MinDistS2P, we show that JLT can be used to further improve the efficacy of ONIAK when the dimension d is not large (say $d \leq 200$) and that JLT significantly outperforms linear scan when d is large. In § 6.3, we evaluate the efficacy of FGoeQF for solving the $\vec{x} \rightarrow \text{vec}(\vec{x}\vec{x}^T)$ dimension blowup problem.

6.1 MinDistS2P Evaluation

6.1.1 Experiment Settings. As described earlier in § 2.1, ONIAK inherits the AMP+(GP-LSH) algorithm used by the ANS (approximate nearest subspace) solution to MinDistS2P proposed in [9, 10]; ONIAK's only improvements over ANS are the use of FGeoQF for reducing the indexing complexity. Although ONIAK and ANS are otherwise the same solution, we believe a quantitative evaluation of ONIAK (and ANS) for MinDistS2P is important for three reasons. First, since [9] was published 15 years ago, neither its ANS algorithm nor its empirical efficacy has been widely known outside the computer vision community, and neither its data nor its code is publicly available. In comparison, this evaluation would assess the efficacy of ONIAK (and ANS [10]) using public datasets, and we will release the code shortly after submitting this revision. Second, in [10], the query efficiency of ANS is measured only on datasets with sizes $n \leq 14,000$, ambient dimensions $d \leq 100$, subspace dimensions $\rho \leq 40$. In comparison, our experiments use larger values of n (up to 10.0M), d (up to 192) and ρ (up to 63). Third, in this work, MinDistS2P is arguably the only natural baby problem to evaluate ONIAK for, since in the other two baby problems evaluated, either the query matrix is degenerative (in ANNS-WL_2) or special workloads have to be used (in ANNS-M).

As stated in § 2.4, when d is moderately large (say between 64 and 200), JLT can be combined with ONIAK (which we call ONIAK+JLT) to achieve even higher ANNS-ALT query efficiency than each solution alone, and when d is large, JLT (alone) is a highly effective solution. In this section, we will also evaluate JLT and ONIAK+JLT to confirm both statements.

6.1.2 Performance Metrics. We measure the query efficiency of ONIAK by *selectivity* [27] (defined in § 5.2.1 and denoted as γ). The query time of ONIAK is roughly equal to that of a linear scan multiplied by γ when n large, since in this case the other steps of MinDistS2P query processing take much less time than checking the ground truth. Hence a smaller γ indicates a better (more effective) LSH solution. To measure the query efficiency of JLT and ONIAK+JLT however, we use query time as the metric, since other steps account for a decent portion of query time in both schemes.

6.1.3 Data and Query Sets. We use four real-life publicly available datasets in this experiment and that for evaluating ANNS-WL_2. They are listed in Table 3. They were intended for evaluating ANNS-L_2 solutions and came with some ANNS-L_2 query vectors (see second last row in Table 3). For each dataset $\mathcal{D}$, we have centered the data vectors in the sense their sample mean is $\vec{0}$.

Table 3: Datasets used in MinDistS2P and ANNS-WL_2.

Dataset	Deep [7]	SIFT [4]	Audio [13]	Trevi [33]
d	96	128	192	4096
n	10.0M	10.0M	52.6K	99.1K
#(L_2-Queries)	10K	10K	200	200
Type	Image	Image	Audio	Image

For each experiment (dataset), we randomly generate 100 query subspaces as follows. First, we uniformly randomly sample $\rho+1$ out of the set of ANNS-L_2 query vectors that accompany the dataset. This sampling is performed without replacement for SIFT and Deep since they came with 10K query vectors each, and with replacement for Audio and Trevi since they came with only 200 query vectors each. Then, we compute the affine subspace generated by these sampled query vectors (aka. the *affine span*) and use it as a query subspace. A query subspace generated this way is highly likely to have the target dimension ρ (and if it does not due to degeneration we scratch it and repeat the sampling process). Since the query subspaces are generated from randomly sampled query vectors, they are expected to be statistically uncorrelated (e.g., do not have the "locality patterns" that exist in ANNS-M workloads).

6.1.4 Experimental Results of ONIAK. We report in Table 4 the selectivity of ONIAK for target recall@50 values between 0.4 and 0.8, on all four datasets except Trevi, whose dimension ($d = 4096$) is too high for ONIAK (and ONIAK+JLT) to handle as explained in §2.4. In each experiment, we tune the number of hash tables to reach the target recall value while keeping other parameters (of GP-LSH and of multi-probe) mostly fixed. These recall values are achieved using between 20 and 100 hash tables. For ease of presenting the experimental results, we use the selectivity-to-recall ratio as the metric, which roughly measures by how much a *partial* LS (e.g., scanning 50% of randomly sampled data vectors to achieve a roughly 50% recall) is slower than ONIAK for achieving the same recall value. Hence, although ONIAK is the sole "contestant" in Table 4, by using this metric, we have indirectly compared ONIAK against the partial LS.

Table 4 shows that ONIAK has fairly good selectivity when the subspace dimension ρ is small. For example, on both Deep and SIFT, when $\rho \leq 7$, the selectivity-to-recall ratio is only between 41.7% and 50.0% when the recall is between 0.4 and 0.5, and is between 49.2% and 66.7% when the recall is between 0.6 and 0.7. The query efficiency gradually degrades as ρ increases. The selectivity-to-recall ratio increases to between 72.3% and 84.0% at $\rho = 15$, and further to between 72.3% and 92.3% at $\rho = 31$. At $\rho = 63$, the selectivity-to-recall ratio is as high as 98.0%, implying that the query efficiency of ONIAK is almost the same as that of linear scan. This trend of query efficiency degrading with ρ can be explained as follows, using the SNR (signal-to-noise ratio) concept introduced in §2.4. As ρ grows larger, the possible "number" of query subspaces (that a universal ONIAK index needs to work with) increases exponentially. However, the (mapped-by-AMP) images of the data vectors are fixed (i.e., do not change with ρ). As a result, when ρ grows larger, it becomes increasingly likely for a mapped (by AMP) query subspace to be very far away (and hence to "feel" roughly equally far way) from all the mapped data vectors (i.e., low SNR).

On Audio, which has a higher dimension of $d = 192$, this trend is more pronounced. When $\rho \leq 3$, the selectivity-to-recall ratio is between 56.0% and 79.3%. It increases to between 66.7% and 85.3% at $\rho = 7$, and further to between 83.3% and 95.8% at $\rho \geq 15$. This is because, the ONIAK index has a lower SNR on Audio (as the SNR scales as $O(1/d)$), and as a result is more vulnerable to the "double SNR whammy" caused by a larger ρ. We do not report results at $\rho \geq 64$ in Table 4, since in this case ONIAK does not perform better than partial LS. Despite the SNR issues, ONIAK achieves fairly good selectivity when ρ is tiny. The selectivity-to-recall ratio is generally less than 60% on Deep and SIFT at $\rho \leq 7$, and less than 70% on Audio at $\rho \leq 3$.

Table 4: Selectivity of ONIAK with recall@50 from 0.4 to 0.8.

Dataset	ρ	0.4	0.5	0.6	0.7	0.8
Deep	1	0.192	0.256	0.316	0.382	0.481
	3	0.167	0.265	0.320	0.381	0.487
	7	0.212	0.300	0.389	0.465	0.570
	15	0.311	0.376	0.451	0.538	0.655
	31	0.321	0.404	0.498	0.588	0.703
	63	0.356	0.475	0.569	0.647	0.747
SIFT	1	0.170	0.233	0.295	0.374	0.468
	3	0.176	0.222	0.301	0.370	0.475
	7	0.233	0.299	0.376	0.467	0.566
	15	0.289	0.382	0.460	0.564	0.672
	31	0.289	0.425	0.503	0.628	0.738
	63	0.353	0.431	0.566	0.681	0.782
Audio	1	0.224	0.281	0.349	0.438	0.609
	3	0.242	0.298	0.397	0.467	0.638
	7	0.311	0.428	0.524	0.561	0.682
	15	0.333	0.447	0.546	0.668	0.762
	31	0.361	0.456	0.568	0.671	0.766

Table 5: Query time (in seconds) with recall@50 $\geq$ 0.8.

Data	ρ	ONIAK+JLT	JLT	LS
Deep	3	0.17+1.41	2.61	7.18
SIFT	3	0.66+1.52	2.88	13.67
Audio	1	0.47+0.012	0.017	0.157
Trevi	127	-	1.83	457.79

6.1.5 Experimental Results of JLT and ONIAK+JLT. Table 5 compares the query efficiencies of ONIAK+JLT, JLT, and (full) LS on these four datasets. The target recall@50 value for both ONIAK+JLT and JLT is 80%, whereas LS achieves 100% recall. In all ONIAK+JLT and JLT experiments, we set the JLT parameter l (the reduced dimension from d) to 15 on Deep and SIFT, to 20 on Audio, and to 40 on Trevi. For a more insightful comparison, for ONIAK+JLT, we break down the query time into (1) the time it takes to hash the query matrix and to read out candidates from the ONIAK hash buckets, which we denote as t_1; and (2) the time it takes to run the JLT filter and to check the ground truth for those that have survived the filtering, which we denote as t_2. Each ONIAK+JLT query time in Table 5 is written as "$t_1 + t_2$".

As shown in Table 5, JLT has much shorter query times than LS: 2.7, 4.7, 9 and 250 times shorter on Deep, SIFT, Audio and Trevi respectively, similar to that reported in §5.3. This outperformance grows roughly linearly with d, because the query time of JLT grows almost linearly with d as explained in §5.3, whereas that of LS grows quadratically with d. As shown in Table 5, ONIAK+JLT further outperforms JLT by 1.65 times on Deep and 1.32 times on SIFT.

This is because in both cases the ONIAK step can *quickly* (much more so than JLT) filter out roughly 50% of the data vectors so that only the remaining 50% or so need to be further vetted by JLT. On Audio, ONIAK+JLT does not outperform JLT because the number of data vectors therein is too small ($n = 52,600$) for the once-per-query and relatively heavy cost of computing the GP-LSH hash values of $g(M) = -\text{vec}(M^T M)$ (where M is the query subspace matrix), which accounts for most of t_1, to be amortized. If Audio were to contain 10M points, ONIAK+JLT would outperform JLT by 1.17 times, assuming ONIAK (with the same or a similar parameterization) can achieve the same selectivity on "10M Audio".

6.2 ANNS-WL_2 Evaluation

In this section, we compare the ANNS-WL_2 query efficiency of ONIAK against that of SL-ALSH [19], the only existing solution (other than LS). Like in §6.1. we measure query efficiency only by selectivity, at recall@50 values ranging from 0.4 to 0.8, since in both ONIAK and SL-ALSH, checking ground truth accounts for most of the query time.

6.2.1 Experiment Settings. For our experiments, we use two datasets sampled uniformly randomly from Deep and SIFT ($n = 10M$ in both as described in Table 3) respectively. Each dataset contains only n=1.0M data vectors, since the SL-ALSH program [18] (implemented in C++) provided by authors of [19] runs out of memory when n is larger. The SL-ALSH program precomputes and stores all possible LSH parameterizations (which bloats its memory consumption), but uses only the optimal one for a given dataset, and hence its query efficiency is not affected by this design choice. We use two types of weights used in [19], namely binary (0 or 1 each with 1/2 probability) and uniform (in $[0, 1]$). Note JLT cannot speed up ONIAK for ANNS-WL_2 even when d is large or moderately large (and hence there is no ONIAK+JLT here), since a ground truth weighted L_2 distance can be computed in $O(d)$ time (rather than $\Omega(d^2)$ time for all other baby problems), and filtering each data vector using JLT takes longer than that.

Table 6: Selectivity when recall@50 ranges from 0.4 to 0.8.

Data	Weight	Name	0.4	0.5	0.6	0.7	0.8
SIFT	Binary	ONIAK	0.309	0.406	0.490	0.606	0.709
		SL-ALSH	0.250	0.330	0.410	0.520	0.640
	Unif.	ONIAK	0.275	0.366	0.458	0.574	0.659
		SL-ALSH	0.160	0.210	0.280	0.365	0.490
Deep	Binary	ONIAK	0.347	0.430	0.520	0.601	0.714
		SL-ALSH	0.290	0.380	0.470	0.575	0.670
	Unif.	ONIAK	0.291	0.383	0.466	0.554	0.661
		SL-ALSH	0.235	0.310	0.395	0.495	0.605

6.2.2 Experiment Results. As shown in Table 6, SL-ALSH has slightly better selectivity than ONIAK in every experiment. For example, on SIFT dataset under uniform weights, SL-ALSH needs to scan between 26% and 43% fewer data vectors than ONIAK. That SL-ALSH outperforms ONIAK is expected, since ONIAK is designed for the general case of the query matrix M being arbitrary, whereas SL-ALSH is designed and optimized for the special and much simpler (computationally) case of M being diagonal.

6.3 Performance Benefits of FGoeQF

In this section, we evaluate by how much times FGoeQF can speed up the computation of an "ONIAK hash value" $h(\text{vec}(\vec{x}\vec{x}^T))$, and can hence reduce the indexing time of ONIAK (used when $d \leq 200$ as explained in § 2.4) and other potential applications (where d can be much larger than 200) such as locality-sensitive teaching described in § 2.2. Table 7 shows the average computation times, for different d values, using the FGoeQF and the naive $O(d^2)$ algorithms respectively. Both algorithms are implemented using the C++ library `Eigen` [17]. Each reported value is the average of between 10^4 and 10^5 runs. Table 7 shows that the FGoeQF algorithm is faster than the naive algorithm by one to three orders of magnitude.

We use two numerical examples to highlight the impact of FGoeQF on the indexing time. In the first example, let $\mathcal{D}$ be a hypothetical dataset containing one billion (1B) 200-dimensional vectors. Suppose we build an ONIAK index on $\mathcal{D}$ using 360 (typical for ONIAK) GP-LSH functions. The indexing time would be roughly 164 days using the naive algorithm and roughly 3 days using FGoeQF, respectively. Note in this case each hash value computation takes the naive algorithm 38.6 µs, but takes ONIAK 0.7 µs. In the second example, let $\mathcal{D}$ be a hypothetical dataset containing 100M 4096-dimensional vectors. Suppose we build an LST index on $\mathcal{D}$ again using 360 GP-LSH functions. The indexing time would be roughly 6,806.8 days (or 18.65 years) using the naive algorithm, and roughly 5.3 days using FGoeQF, respectively.

Table 7: Computation time (in µs) per GP-LSH hashing.

$d =$	32	64	128	256	512	1024	4096
FGoeQF	0.094	0.19	0.35	0.7	1.6	2.8	12.7
Naive	0.991	3.94	15.77	63.3	252.0	1011.8	16,336.3

7 CONCLUSION

In this work, we formulate ANNS-ALT (and its dual ANNS-ALTD), which are mother problems to a wide range of ANNS problems with important applications. We propose the ONIAK solution, which answers all future ANNS-ALT queries using a single universal index and zero re-indexing overhead, and the JLT solution, which performs well on large dimensions. We also propose the FGoeQF technique, which solves the fallout of the long-standing dimension blowup problem and as a result significantly reduces the indexing time. Finally, we show by experiments that ONIAK has better query efficiency than linear scan, and JLT has much better query efficiency than any competitor for all baby problems except ANNS-WL_2.

Acknowledgment. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1909048, CNS-2007006, and CNS-2051800. We thank the reviewers for their suggestions that have greatly improved the quality and the readability of this paper.

REFERENCES

[1] Nasir Ahmed and Kamisetty Ramamohan Rao. 1975. *Walsh-Hadamard Transform.* Springer Berlin Heidelberg, Berlin, Heidelberg, 99–152. https://doi.org/10.1007/978-3-642-45450-9_6

[2] Nir Ailon and Bernard Chazelle. 2009. The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors. *SIAM J. Comput.* 39, 1 (2009), 302–322. https://doi.org/10.1137/060673096 arXiv:https://doi.org/10.1137/060673096

[3] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter. Walter. 2002. *Molecular Biology of the Cell* (4 ed.). Garland Science. https://www.ncbi.nlm.nih.gov/books/NBK21054/

[4] Laurent Amsaleg and Hervé Jégou. 2010. Datasets for ANN neighbor search. http://corpus-texmex.irisa.fr/.

[5] Greg W. Anderson, Alice Guionnet, and Ofer Zeitouni. 2009. *Real and complex Wigner matrices.* Cambridge University Press, Cambridge, UK, 6–89. https://doi.org/10.1017/CBO9780511801334.003

[6] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and Optimal LSH for Angular Distance. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15).* MIT Press, Cambridge, MA, USA, 1225–1233. http://dl.acm.org/citation.cfm?id=2969239.2969376

[7] Artem Babenko and Victor Lempitsky. [n.d.]. Deep: Datasets of deep descriptors. http://sites.skoltech.ru/compvision/noimi/.

[8] Pierre Baldi, Kyle Cranmer, Taylor Faucett, Peter Sadowski, and Daniel Whiteson. 2016. HEPMASS Data Set. http://archive.ics.uci.edu/ml/datasets/hepmass.

[9] Ronen Basri, Tal Hassner, and Lihi Zelnik-Manor. 2007. Approximate Nearest Subspace Search with Applications to Pattern Recognition. In *2007 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, 1–8. https://ieeexplore.ieee.org/document/4270226

[10] Ronen Basri, Tal Hassner, and Lihi Zelnik-Manor. 2011. Approximate Nearest Subspace Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 2 (2011), 266–278. https://doi.org/10.1109/TPAMI.2010.110

[11] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Francois Fagan, Cedric Gouy-Pailler, Anne Morvan, Nouri Sakr, Tamas Sarlos, and Jamal Atif. 2017. Structured adaptive and random spinners for fast machine learning computations. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Aarti Singh and Jerry Zhu (Eds.), Vol. 54. PMLR, Ft. Lauderdale, FL, USA, 1020–1029. https://proceedings.mlr.press/v54/bojarski17a.html

[12] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive Hashing Scheme Based on P-stable Distributions. In *Proceedings of the Annual Symposium on Computational Geometry (SCG '04).* ACM, New York, NY, USA, 253–262. https://doi.org/10.1145/997817.997857

[13] Princeton University Computer Science Department. [n.d.]. Audio Dataset. http://www.cs.princeton.edu/cass/audio.tar.gz.

[14] Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. 2005. Neighbourhood Components Analysis. In *Advances in Neural Information Processing Systems*, L. Saul, Y. Weiss, and L. Bottou (Eds.), Vol. 17. MIT Press, Vancouver, BC, Canada. https://proceedings.neurips.cc/paper/2004/file/42fe880812925e520249e808937738d2-Paper.pdf

[15] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 9, 1 (Sept. 2015), 1–12. https://doi.org/10.14778/2850469.2850470

[16] Qiang Huang, Yifan Lei, and Anthony K. H. Tung. 2021. *Point-to-Hyperplane Nearest Neighbor Search Beyond the Unit Hypersphere.* Association for Computing Machinery, New York, NY, USA, 777–789. https://doi.org/10.1145/3448016.3457240

[17] Benoît Jacob, Gaël Guennebaud, and et al. n.d.. Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. https://eigen.tuxfamily.org.

[18] Yifan Lei. 2020. Sublinear Time Nearest Neighbor Search over Generalized Weighted Space. https://github.com/1flei/aws_alsh.

[19] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony Tung. 2019. Sublinear Time Nearest Neighbor Search over Generalized Weighted Space. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, Long Beach, CA, USA, 3773–3781. https://proceedings.mlr.press/v97/lei19a.html

[20] Christophe Leys, Olivier Klein, Yves Dominicy, and Christophe Ley. 2018. Detecting Multivariate Outliers: Use a Robust Variant of the Mahalanobis Distance. *Journal of Experimental Social Psychology* 74 (2018), 150–156. https://doi.org/10.1016/j.jesp.2017.09.011

[21] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. 2019. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1–1. https://doi.org/10.1109/TKDE.2019.2909204

[22] Weiyang Liu, Bo Dai, Ahmad Humayun, Charlene Tay, Chen Yu, Linda B. Smith, James M. Rehg, and Le Song. 2017. Iterative Machine Teaching. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, 2149–2158. https://proceedings.mlr.press/v70/liu17b.html

[23] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07).* VLDB Endowment, Vienna, Austria, 950–961. http://dl.acm.org/citation.cfm?id=1325851.1325958

[24] Carl D. Meyer. 2000. *Matrix Analysis and Applied Linear Algebra.* Vol. 71. Siam, Philadelphia, PA, USA.

[25] Ilya Razenshteyn and Ludwig Schmidt. 2017. FALCONN - FAst Lookups of Cosine and Other Nearest Neighbors. https://github.com/FALCONN-LIB/FALCONN.

[26] Jie Ren, Minjia Zhang, and Dong Li. 2020. HM-ANN: Efficient Billion-Point Nearest Neighbor Search on Heterogeneous Memory. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 10672–10684. https://proceedings.neurips.cc/paper/2020/file/788d986905533aba051261497ecffcbb-Paper.pdf

[27] Kexin Rong, Clara E. Yoon, Karianne J. Bergen, Hashem Elezabi, Peter Bailis, Philip Levis, and Gregory C. Beroza. 2018. Locality-sensitive Hashing for Earthquake Detection: A Case Study of Scaling Data-driven Science. 11, 11 (July 2018), 1674–1687. https://doi.org/10.14778/3236187.3236214

[28] Peter M. Roth, Martin Hirzer, Martin Köstinger, Csaba Beleznai, and Horst Bischof. 2014. *Mahalanobis Distance Learning for Person Re-identification.* Springer London, London, 247–267. https://doi.org/10.1007/978-1-4471-6296-4_12

[29] scikit-learn developers. n.d.. The make classification method. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html.

[30] Gilbert Strang. 2016. *Introduction to Linear Algebra.* Vol. 5. Wellesley - Cambridge, Wellesley, MA.

[31] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: Solving C-approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. *PVLDB* 8, 1 (2014), 1–12. https://doi.org/10.14778/2735461.2735462

[32] Sudheendra Vijayanarasimhan, Prateek Jain, and Kristen Grauman. 2014. Hashing Hyperplane Queries to Near Points with Applications to Large-Scale Active Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 2 (2014), 276–288. https://doi.org/10.1109/TPAMI.2013.121

[33] Simon Winder, Matt Brown, Noah Snavely, Steven Seitz, and Richard Szeliski. [n.d.]. Trevi: Local Image Descriptors Data. http://phototour.cs.washington.edu/patches/default.htm.

[34] Shiming Xiang, Feiping Nie, and Changshui Zhang. 2008. Learning a Mahalanobis Distance Metric for Data Clustering and Classification. *Pattern Recogn.* 41, 12 (Dec. 2008), 3600–3612. https://doi.org/10.1016/j.patcog.2008.05.018

[35] Zhaozhuo Xu, Beidi Chen, Chaojian Li, Weiyang Liu, Le Song, Yingyan Lin, and Anshumali Shrivastava. 2021. Locality Sensitive Teaching. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 18049–18062. https://proceedings.neurips.cc/paper/2021/file/95c3f1a8b262ec7a929a8739e21142d7-Paper.pdf

[36] Zhaozhuo Xu, Beidi Chen, Chaojian Li, Weiyang Liu, Le Song, Yingyan Lin, and Anshumali Shrivastava. 2021. Locality Sensitive Teaching. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 18049–18062. https://proceedings.neurips.cc/paper/2021/file/95c3f1a8b262ec7a929a8739e21142d7-Paper.pdf

[37] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-Ranging LSH for Maximum Inner Product Search. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/b60c5ab647a27045b462934977ccad9a-Paper.pdf

[38] Fuzhen Zhang. 2011. *Matrix Theory: Basic Results and Techniques.* Springer, New York, NY. https://doi.org/10.1007/978-1-4614-1099-7_7

[39] Huayi Zhang, Lei Cao, Yizhou Yan, Samuel Madden, and Elke A. Rundensteiner. 2020. Continuously Adaptive Similarity Search. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20).* Association for Computing Machinery, New York, NY, USA, 2601–2616. https://doi.org/10.1145/3318464.3380601

[40] Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S. Jensen. 2020. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *Proc. VLDB Endow.* 13, 5 (Jan. 2020), 643–655. https://doi.org/10.14778/3377369.3377374

[41] Lei Zhou, Xiao Bai, Xianglong Liu, Jun Zhou, and Edwin R. Hancock. 2020. Learning Binary Code for Fast Nearest Subspace Search. *Pattern Recognition* 98 (2020), 107040. https://doi.org/10.1016/j.patcog.2019.107040