LoRANN: Low-Rank Matrix Factorization for Approximate Nearest Neighbor Search

Elias Jääsaari [†] **Ville Hyvönen** [‡] **Teemu Roos** [†]
[†]Department of Computer Science, University of Helsinki
[‡]Department of Computer Science, Aalto University
`{elias.jaasaari,teemu.roos}@helsinki.fi; ville.2.hyvonen@aalto.fi`

Abstract

Approximate nearest neighbor (ANN) search is a key component in many modern machine learning pipelines; recent use cases include retrieval-augmented generation (RAG) and vector databases. Clustering-based ANN algorithms, that use score computation methods based on product quantization (PQ), are often used in industrial-scale applications due to their scalability and suitability for distributed and disk-based implementations. However, they have slower query times than the leading graph-based ANN algorithms. In this work, we propose a new supervised score computation method based on the observation that inner product approximation is a multivariate (multi-output) regression problem that can be solved efficiently by reduced-rank regression. Our experiments show that on modern high-dimensional data sets, the proposed reduced-rank regression (RRR) method is superior to PQ in both query latency and memory usage. We also introduce `LoRANN`[1], a clustering-based ANN library that leverages the proposed score computation method. `LoRANN` is competitive with the leading graph-based algorithms and outperforms the state-of-the-art GPU ANN methods on high-dimensional data sets.

1 Introduction

In modern machine learning applications, data is often stored as embeddings, i.e., as vectors in a high-dimensional vector space where representations of semantically similar items are close to each other. Consequently, similarity search in high-dimensional vector spaces is a key algorithmic primitive used in many pipelines, such as semantic search engines and recommendation systems. Since the data sets are usually both large and high-dimensional, *approximate* nearest neighbor (ANN) search is deployed to speed up similarity search in many applications (Li et al., 2019).

Recent use cases of ANN search include retrieval-augmented generation (RAG) (Lewis et al., 2020; Guu et al., 2020; Borgeaud et al., 2022; Shi et al., 2024) and approximate attention computation in Transformer-based architectures (Kitaev et al., 2020; Vyas et al., 2020; Roy et al., 2021). ANN search is also a key operation in vector databases that are used to store embeddings for industrial-scale applications (see, e.g., Wang et al., 2021; Guo et al., 2022; Pan et al., 2024).

The state-of-the-art methods for ANN search can be classified into clustering-based and graph-based algorithms (for a recent survey, see Bruch, 2024). In the comprehensive ANN benchmark (Aumüller et al., 2020), the leading graph algorithms HNSW (Malkov and Yashunin, 2018) and NGT (Iwasaki and Miyazaki, 2018) have faster query times than clustering-based algorithms. However, clustering-based algorithms are often used in industrial-scale applications (see, e.g., Chen et al., 2021; Douze et al., 2024) due to their smaller memory footprints and faster index construction times. They are

[1] `https://github.com/ejaasaari/lorann`

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

also suitable for distributed implementations and hybrid solutions that use persistent storage such as SSDs or blob storage (Chen et al., 2021; Gottesbüren et al., 2024).

The key components of clustering-based algorithms are *clustering* and *score computation*. In the indexing phase, the corpus, i.e., the data set from which the nearest neighbors are searched, is partitioned via clustering. In the query phase, w clusters are selected and the corpus points that belong to the selected clusters are scored. The points with the highest scores are selected into a candidate set that can be further re-ranked. The state-of-the-art clustering-based algorithms (Jegou et al., 2011; Guo et al., 2020; Sun et al., 2023) use variants of *product quantization* (PQ) (Jegou et al., 2011) with highly optimized implementations (see, e.g., André et al., 2015) for score computation.

In this article, we propose a supervised score computation method that improves the query latency of clustering-based ANN methods, making them competitive with the leading graph algorithms. Our key observation is that estimating the dissimilarities between the query point and the cluster points is a multivariate (multi-output) regression problem. For the most common dissimilarity measures, it is sufficient to estimate the inner products between the query point and the cluster points, and the ordinary least squares (OLS) estimate is the exact solution for this regression problem. The proposed method approximates the OLS solution by reduced-rank regression (Izenman, 1975). We further approximate the reduced-rank regression estimates using 8-bit integer computations, improving query latency and memory consumption. Reduced-rank regression (RRR) is a simpler method than PQ, and our experimental results show that it is faster at any given recall level and memory usage.

To make our work available for practical applications of ANN search, we introduce `LoRANN`, a clustering-based ANN library that leverages the proposed score computation method. Since the low memory usage and the simple computational structure of reduced-rank regression make it well-suited for GPUs, `LoRANN` also includes a GPU implementation of the proposed method.

In summary, our contributions are:

- We propose reduced-rank regression (RRR) as a new supervised score computation method for clustering-based ANN search (see Section 3).
- We verify experimentally that RRR outperforms PQ both at the optimal hyperparameters (Section 7.1.1) and at fixed memory consumption (Section 7.1.2), and that it naturally adapts to the query distribution in the out-of-distribution (OOD) setting (Section 5).
- We introduce `LoRANN`, a clustering-based ANN library that contains efficient CPU and GPU implementations of the proposed score computation method (Section 4).
- We show that `LoRANN` outperforms the leading clustering-based libraries Faiss (Douze et al., 2024) and ScaNN (Guo et al., 2020), and has faster query times than the leading graph-based library GLASS at recall levels under 90% on most data sets (Section 7.2.2). `LoRANN` outperforms the SOTA GPU methods on high-dimensional data (Section 7.2.3).

2 Background

In this section, we first review the notation of approximate nearest neighbor (ANN) search and then describe the standard structure of clustering-based ANN algorithms.

2.1 ANN search

Let $\mathbf{x} \in \mathbb{R}^d$ be a query point, and let $\{\mathbf{c}_j\}_{j=1}^m \subset \mathbb{R}^d$ be the *corpus*, i.e., the set of points from which the nearest neighbors are retrieved. The k nearest neighbors of $\mathbf{x}$ are defined as

$$\textsc{nn}_k(\mathbf{x}; \{\mathbf{c}_j\}_{j=1}^m, d) := \{j \in [m] \,:\, d(\mathbf{x}, \mathbf{c}_j) \leq d(\mathbf{x}, \mathbf{c}_{(k)})\}, \tag{1}$$

where $d : \mathbb{R}^d \to \mathbb{R}$ is a dissimilarity measure, $\mathbf{c}_{(1)}, \dots, \mathbf{c}_{(m)}$ denote the corpus points that are ordered in ascending order w.r.t. their dissimilarity to the query point $\mathbf{x}$, and $[m] := \{1, \dots, m\}$.

Commonly used dissimilarity measures are the Euclidean distance $d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_2$, the (negative) inner product $d(\mathbf{a}, \mathbf{b}) = -\langle \mathbf{a}, \mathbf{b} \rangle$, and the cosine (angular) distance $d(\mathbf{a}, \mathbf{b}) = 1 - \langle \mathbf{a}/\|\mathbf{a}\|_2, \mathbf{b}/\|\mathbf{b}\|_2 \rangle$. The special case where the dissimilarity measure is the negative inner product is often called maximum inner product search (MIPS) (see, e.g., Guo et al., 2020; Lu et al., 2023; Zhao et al., 2023). In ANN search, the exact solution $\textsc{nn}_k(\mathbf{x}; \{\mathbf{c}_j\}_{j=1}^m, d)$ is approximated.

The effectiveness of ANN algorithms is typically measured by *recall*, i.e., the fraction of the true k nearest neighbors returned by the algorithm. The efficiency is measured by query latency or, equivalently, by queries per second (QPS) (see, e.g., Li et al., 2019; Aumüller et al., 2020).

2.2 Clustering-based ANN search

Partitioning-based ANN algorithms, such as tree-based (e.g., Muja and Lowe, 2014; Dasgupta and Sinha, 2015; Jääsaari et al., 2019) and hashing-based algorithms (e.g., Charikar, 2002; Weiss et al., 2008; Liu et al., 2011), build an index by partitioning the corpus into L elements. In the query phase, they use a routing function $\tau : \mathbb{R}^d \to [L]^w$ to assign the query point into w partition elements.

Clustering-based ANN algorithms (see, e.g., Bruch, 2024, Chapter 7) are partitioning-based ANN algorithms that partition the corpus via clustering. ANN indexes based on clustering are also often called *inverted file* (IVF) indexes (Jegou et al., 2011). The most commonly used clustering method is k-means clustering, specifically standard k-means when the dissimilarity measure d is the Euclidean distance, and spherical k-means (Dhillon and Modha, 2001) when d is the inner product or the cosine distance. While there exists recent exploratory work on alternative query routing methods (Gottesbüren et al., 2024; Vecchiato et al., 2024; Bruch et al., 2024), the most common method is centroid-based routing where, for the set $\{\boldsymbol{\mu}_l\}_{l=1}^L$ of cluster centroids,

$$\tau(\mathbf{x}) = \text{NN}_w(\mathbf{x}; \{\boldsymbol{\mu}_l\}_{l=1}^L, d),$$

i.e., the w clusters whose centroids are closest to the query point are selected. We follow this standard practice by using k-means clustering with centroid-based routing. However, the proposed score computation method can be combined with any partitioning and routing method.

After routing, the corpus points that belong to the selected w clusters are scored (see Section 6 for a discussion of score computation methods), and the t highest scoring points are selected into the candidate set. This candidate set can be re-ranked by evaluating the true dissimilarities between $\mathbf{x}$ and the candidate set points. Finally, the k most similar points are returned as the approximate k-nn.

3 Reduced-rank regression

In this section, we derive the proposed supervised score computation method. First, we formulate dissimilarity approximation as a multivariate regression problem. We then show how the exact OLS solution to this problem can be approximated by reduced-rank regression (RRR). Finally, we show how RRR can be implemented efficiently using 8-bit integer vector-matrix multiplications.

3.1 Dissimilarity approximation as a multivariate regression problem

We consider the task of approximating the dissimilarities $d(\mathbf{x}, \mathbf{c}_j)$ between the query point $\mathbf{x}$ and the corpus points $\mathbf{c}_j$ that belong to the lth cluster. Denote the set of the indices of these corpus points by I_l, their number by $m_l := |I_l|$, and the matrix containing them as rows by $\mathbf{C}_l \in \mathbb{R}^{m_l \times d}$. In what follows, to avoid cluttering the notation, we drop the subscript l denoting the cluster from matrices, e.g., we denote $\mathbf{C}_l$ by $\mathbf{C}$. We also assume, w.l.o.g., that the corpus is indexed so that $I_l = \{1, \dots, m_l\}$. This task can now be formulated as a multivariate regression problem where the output is defined as a $1 \times m_l$ matrix $\mathbf{y} = [y_1 \cdots y_{m_l}]$, where $y_j = d(\mathbf{x}, \mathbf{c}_j)$ for each $j = 1, \dots, m_l$.

We consider the cases where d is the (negative) inner product, the Euclidean distance, or the cosine (angular) distance. In all three cases, it is sufficient to estimate the inner products. For Euclidean distance, $\arg\min_{j \in I_l} \|\mathbf{x} - \mathbf{c}_j\|_2 = \arg\min_{j \in I_l}(-2\mathbf{x}^T\mathbf{c}_j + \|\mathbf{c}_j\|_2^2)$, where the norms $\|\mathbf{c}_j\|_2$ can be precomputed. For cosine distance, $\arg\min_{j \in I_l}(1 - \cos(\mathbf{x}, \mathbf{c}_j)) = \arg\min_{j \in I_l}(-\mathbf{x}^T\mathbf{c}_j)$ if the corpus points are normalized to have unit norm.

3.2 Reduced-rank regression solution

We approximate the exact solution $\mathbf{y} = \mathbf{x}^T\mathbf{C}^T$ of the regression problem defined in the previous section by a low-rank approximation. We assume the standard supervised learning setting, i.e., that we have a sample $\{\mathbf{x}_i\}_{i=1}^n$ from the query distribution $\mathcal{Q}$ (the corpus can also be used as the training set if no separate training set is available). To train the lth model, we use all the training set points that are routed into the lth cluster. When the standard centroid-based routing is used, these are the training set points that have $\boldsymbol{\mu}_l$, i.e., the centroid of the lth cluster, among their w closest centroids. Denote the set of indices of these training set points by $J_l := \{i \in [n] \,:\, l \in \tau(\mathbf{x}_i)\}$, and their number by $n_l := |J_l|$. The output values of the training set of the lth model are given by $\mathbf{Y} := \mathbf{X}\mathbf{C}^T \in \mathbb{R}^{n_l \times m_l}$, where we denote by $\mathbf{X} \in \mathbb{R}^{n_l \times d}$ the matrix containing the training set points $\{\mathbf{x}_i\}_{i \in J_l}$ as rows.

To approximate the dissimilarities between the query point $\mathbf{x}$ and the cluster points $\{\mathbf{c}_j\}_{j \in I_l}$, we consider the linear model $\mathbf{x}^T\boldsymbol{\beta}$, where $\boldsymbol{\beta} \in \mathbb{R}^{d \times m_l}$ is a matrix containing the parameters of the model, and minimize the mean squared error $\mathbb{E}_{\mathbf{x}\sim\mathcal{Q}}[\|\mathbf{y} - \mathbf{x}^T\boldsymbol{\beta}\|_2^2 \, \mathbb{I}\{l \in \tau(\mathbf{x})\}]$ (the indicator function selects the query points that routed into the lth cluster). The unconstrained least squares solution $\hat{\boldsymbol{\beta}}_{\text{OLS}} = \mathbf{C}^T$ reproduces the exact inner products $\mathbf{y} = \mathbf{x}^T\mathbf{C}^T$. In order to reduce the computational complexity of evaluating the model predictions, we constrain the rank of the parameter matrix: $\text{rank}(\boldsymbol{\beta}) \leq r < \min(d, m_l)$. Under this constraint, the parameter matrix can be written using a low-rank matrix factorization $\boldsymbol{\beta} = \mathbf{A}\mathbf{B}$, where $\mathbf{A} \in \mathbb{R}^{d \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times m_l}$, and, consequently, the model predictions $\hat{\mathbf{y}} = (\mathbf{x}^T\mathbf{A})\mathbf{B}$ can be computed with $\Theta(r(d + m_l))$ operations. When the rank r is sufficiently low, this is significantly faster than computing the exact inner products which requires $\Theta(dm_l)$ operations. Our experiments (Section 7) indicate that fixing this hyperparameter to $r = 32$ works well with a wide range of data sets encompassing dimensionalities between 128 and 1536.

The optimal low-rank solution can be found by minimizing the training loss

$$\hat{\boldsymbol{\beta}}_{\text{RRR}} = \underset{\boldsymbol{\beta}\,:\,\text{rank}(\boldsymbol{\beta}) \leq r}{\arg\min} \; \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_F^2,$$

where $\|\cdot\|_F$ is the Frobenius norm. This is the well-known *reduced-rank regression* problem (Izenman, 1975). Denote the singular value decomposition (SVD) of $\mathbf{Y}$ as $\mathbf{Y} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$, where $\boldsymbol{\Sigma}$ is a non-negative diagonal matrix and $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices. The standard reduced-rank regression solution is $\hat{\boldsymbol{\beta}}_{\text{RRR}} = \hat{\boldsymbol{\beta}}_{\text{OLS}}\mathbf{V}_r\mathbf{V}_r^T = \mathbf{C}^T\mathbf{V}_r\mathbf{V}_r^T = \mathbf{A}\mathbf{B}$, where $\mathbf{V}_r \in \mathbb{R}^{m_l \times r}$ denotes the matrix that contains the first r columns of $\mathbf{V}$ (i.e., the first r right singular vectors of the least squares fit $\mathbf{Y} = \mathbf{X}\mathbf{C}^T$) , $\mathbf{A} := \mathbf{C}^T\mathbf{V}_r$, and $\mathbf{B} := \mathbf{V}_r^T$. In practice, we use a fast randomized algorithm (Halko et al., 2011) to compute only $\mathbf{V}_r$ instead of the full SVD. Observe that the reduced-rank regression solution is different from the most obvious low-rank matrix factorization of the OLS solution computed via an SVD of the matrix $\mathbf{C}$ (see Appendix A).

3.3 8-bit quantization

The simple computational structure of the reduced-rank regression solution enables us to further improve its query latency and memory consumption by using integer quantization. For each cluster, we quantize the matrices $\mathbf{A}$ and $\mathbf{B}$ to 8-bit integer precision. By also quantizing the query vector $\mathbf{x}$, the model prediction $\hat{\mathbf{y}} = \mathbf{x}^T\hat{\boldsymbol{\beta}}_{\text{RRR}} = (\mathbf{x}^T\mathbf{A})\mathbf{B}$ can be computed efficiently in two 8-bit integer vector-matrix products. We use *absmax quantization* (e.g., Dettmers et al., 2022), where the elements of a vector $\mathbf{x}$ are scaled to the range $[-127, 127]$ by multiplying with a constant $c_{\mathbf{x}}$ such that $\mathbf{x}_{i8} = \lfloor(127/\|\mathbf{x}_{f32}\|_\infty) \cdot \mathbf{x}_{f32}\rceil = \lfloor c_{\mathbf{x}}\mathbf{x}_{f32}\rceil$, where $\lfloor\cdot\rceil$ denotes rounding to the nearest integer.

We quantize the matrices $\mathbf{A}$ and $\mathbf{B}$ by applying absmax quantization to each column of the given matrix, resulting in vectors $\mathbf{c}_{\mathbf{A}}$ and $\mathbf{c}_{\mathbf{B}}$ of scaling constants. We can then recover a 32-bit floating-point approximation to the vector-matrix product $\mathbf{r} = \mathbf{x}^T\mathbf{A}$ with

$$\mathbf{r}_{f32} \approx \frac{1}{c_{\mathbf{x}}}\mathbf{r}_{i32} \oslash \mathbf{c}_{\mathbf{A}} =: \mathbf{s} \odot \mathbf{r}_{i32} = \mathbf{s} \odot \mathbf{x}_{i8}^T\mathbf{A}_{i8} = \mathbf{s} \odot Q(\mathbf{x})^T Q(\mathbf{A}_{f32}),$$

where $Q(\cdot)$ denotes absmax quantization, and $\oslash$ and $\odot$ denote element-wise division and multiplication, respectively. To compute $\hat{\mathbf{y}} = \mathbf{r}^T\mathbf{B}$ in the same fashion, we can first re-quantize $\mathbf{r}$.

To ensure minimal loss of precision from the quantization, we rotate $\mathbf{x}$ before quantization by multiplying $\mathbf{x}$ with a random rotation matrix; this spreads the variance among the dimensions of $\mathbf{x}$. Similarly, we rotate the vector $\mathbf{r}$ resulting from the first product $\mathbf{r} = \mathbf{x}^T\mathbf{A}$ before re-quantization. Since $\mathbf{A} = \mathbf{C}^T\mathbf{V}_r$ and $\mathbf{B} = \mathbf{V}_r^T$, we can rotate $\mathbf{r}$ by rotating $\mathbf{V}_r$ beforehand at no extra cost.

Memory usage Storing each matrix $\mathbf{A}_{i8} \in [\mathbb{Z}]_{256}^{d \times r}$ takes dr bytes and each matrix $\mathbf{B}_{i8} \in [\mathbb{Z}]_{256}^{r \times m_l}$ takes rm_l bytes. Thus in a clustering-based ANN index with L clusters and m corpus points, the total memory consumption of RRR is of order $Ldr + rm$ bytes. In our experiments (Section 7), we use $r = 32$ for all data sets, while L is typically of order $\sqrt{m}$.

4 LoRANN

In this section, we describe the additional implementation details of `LoRANN`, an open-source library that combines the standard template of clustering-based ANN search described in Section 2.2 with the score computation method described in Section 3. Using dimensionality reduction is particularly efficient for RRR (Section 4.1) and works well with 8-bit integer quantization (Section 4.2). Finally, we describe the GPU implementation of `LoRANN` (Section 4.3).

4.1 Dimensionality reduction

With a moderate-sized corpus and high-dimensional data, computing the first vector-matrix product of the model prediction $\hat{\mathbf{y}} = (\mathbf{x}^T\mathbf{A})\mathbf{B}$ can be more expensive than the second. In `LoRANN`, we further approximate the product by first projecting the query into a lower-dimensional space. We use the projection matrix $\mathbf{W}_s \in \mathbb{R}^{d\times s}$ whose columns are the first s eigenvectors of $\mathbf{X}_{\text{global}}^T\mathbf{X}_{\text{global}}$, where $\mathbf{X}_{\text{global}} \in \mathbb{R}^{n\times d}$ is the matrix containing the training set points $\{\mathbf{x}_i\}_{i=1}^n$. To estimate the reduced-rank regression models, we use the s-dimensional approximations $\tilde{\mathbf{x}}_i = \mathbf{W}_s^T\mathbf{x}_i$ as inputs, but the true inner products $\mathbf{x}_i^T\mathbf{c}_j$ as outputs. In this case, the reduced-rank regression estimate of the lth model is $\hat{\boldsymbol{\beta}}_{\text{RRR}} = \hat{\boldsymbol{\beta}}_{\text{OLS}}\mathbf{V}_r\mathbf{V}_r^T = (\mathbf{X}\mathbf{W}_s)^\dagger\mathbf{Y}\mathbf{V}_r\mathbf{V}_r^T \in \mathbb{R}^{s\times m_l}$, where $\hat{\boldsymbol{\beta}}_{\text{OLS}} := (\mathbf{X}\mathbf{W}_s)^\dagger\mathbf{Y}$ is the full-rank solution, and $\mathbf{V}_r \in \mathbb{R}^{m_l\times r}$ is the matrix whose columns are the first r right singular vectors of $\mathbf{Y}$. Thus, now $\mathbf{A} := (\mathbf{X}\mathbf{W}_s)^\dagger\mathbf{Y}\mathbf{V}_r \in \mathbb{R}^{s\times r}$ and $\mathbf{B} := \mathbf{V}_r^T \in \mathbb{R}^{r\times m_l}$.

We observe that computing query-to-centroid distances in the s-dimensional space yields a minor performance improvement. In the indexing phase, we perform k-means clustering using $\tilde{\mathbf{c}}_j = \mathbf{W}_s^T\mathbf{c}_j$. In the query phase, the s-dimensional approximation of the query point, $\tilde{\mathbf{x}} = \mathbf{W}_s^T\mathbf{x}$, is used to compute the distances to the cluster centroids and the predictions $\hat{\mathbf{y}} = \tilde{\mathbf{x}}^T\hat{\boldsymbol{\beta}}_{\text{RRR}}$. The original d-dimensional query point $\mathbf{x}$ is used for the dissimilarity evaluations in the final re-ranking step.

4.2 Quantization implementation

The dimensionality reduction works particularly well with the 8-bit integer quantization described in Section 3.3. After dimensionality reduction, the first component of $\tilde{\mathbf{x}}$ corresponds to the principal axis. Thus, to further reduce the precision lost by quantization, we employ a mixed-precision decomposition by not quantizing the first component of $\tilde{\mathbf{x}}$ and the first row of both $\mathbf{A}$ and $\mathbf{B}$. Moreover, by premultiplying the projection matrix $\mathbf{W}_s$ with a random rotation matrix, we rotate the query point $\tilde{\mathbf{x}}$ at no extra cost before quantization. For re-quantizing $\mathbf{r} = \tilde{\mathbf{x}}^T\mathbf{A}$, since $\mathbf{A} = (\mathbf{X}\mathbf{W}_s)^\dagger\mathbf{Y}\mathbf{V}_r$ and $\mathbf{B} = \mathbf{V}_r^T$, we can again randomly rotate $\mathbf{V}_r$ beforehand at no extra cost.

We compute the 8-bit vector-matrix products $\mathbf{r}_{i32} = \tilde{\mathbf{x}}_{i8}^T\mathbf{A}_{i8}$ and $\hat{\mathbf{y}}_{i32} = \mathbf{r}_{i8}^T\mathbf{B}_{i8}$ efficiently on modern CPUs using VPDPBUSD instructions in the AVX-512 VNNI instruction set.[2] Since a VPDPBUSD instruction computes dot products of signed 8-bit integer vectors and unsigned 8-bit integer vectors, we store $\mathbf{A}_{i8}$ and $\mathbf{B}_{i8}$ as unsigned 8-bit integer matrices $\mathbf{A}'_{i8} = \mathbf{A}_{i8} + 128 \cdot \mathbf{1}_{s\times r} \in [\mathbb{Z}]_{256}^{d\times r}$ and $\mathbf{B}'_{i8} = \mathbf{B}_{i8} + 128 \cdot \mathbf{1}_{r\times m} \in [\mathbb{Z}]_{256}^{r\times m}$, and compute $\mathbf{r}_{i32} = \tilde{\mathbf{x}}_{i8}^T\mathbf{A}_{i8} = \tilde{\mathbf{x}}_{i8}^T\mathbf{A}'_{i8} - 128 \cdot \tilde{\mathbf{x}}_{i8}^T\mathbf{1}_{s\times r}$. The dimensionality reduction lowers the memory usage of `LoRANN` from $Ldr + rm$ to $Lsr + rm$ bytes.

4.3 GPU implementation

Hardware accelerators such as GPUs and TPUs can be used to speed up ANN search for queries that arrive in batches (Johnson et al., 2019; Zhao et al., 2020; Groh et al., 2022; Ootomo et al., 2023). The computational structure of RRR, consisting of vector-matrix multiplications, makes it easy to implement `LoRANN` for accelerators, and the low memory usage of RRR (see Section 7.1.2) makes it ideal for accelerators that typically have a limited amount of memory.

Given a query matrix $\mathbf{Q} \in \mathbb{R}^{|Q|\times d}$ with $|Q|$ queries, we need to compute the products $\mathbf{q}_i^T\mathbf{A}_{il}\mathbf{B}_{il}$ for all $i = 1, \ldots, |Q|$ and $l = 1, \ldots, w$. Here we denote by $\mathbf{A}_{il}$ and $\mathbf{B}_{il}$ the matrices $\mathbf{A}$ and $\mathbf{B}$ of the lth cluster in the set of w clusters the ith query point is routed into. We compute the required products efficiently using one batched matrix multiplication $\mathbf{Q}_\mathbf{T}\mathbf{A}_\mathbf{T}\mathbf{B}_\mathbf{T}$ by representing $\mathbf{Q}$ as a $|Q| \times 1 \times 1 \times d$ tensor $\mathbf{Q}_\mathbf{T}$, the matrices $\mathbf{A}_{il}$ as one $|Q| \times w \times s \times r$ tensor $\mathbf{A}_\mathbf{T}$, and the matrices

[2] `https://en.wikichip.org/wiki/x86/avx512_vnni`

$\mathbf{B}_{il}$ as one $|Q| \times w \times r \times M$ tensor $\mathbf{B_T}$, where M is the maximum number of points in a single cluster. To avoid inefficiencies due to padding clusters with fewer than M points, we use an efficient balanced k-means algorithm (de Maeyer et al., 2023) to ensure that the clusters are balanced such that the maximum difference in cluster sizes is $\Delta = 16$.

On GPUs, 8-bit integer multiplication is presently both less efficient and less supported than 16-bit floating-point multiplication. Therefore, we use 16-bit floating-point numbers to perform all computations on a GPU. However, the 8-bit quantization scheme can still be useful on accelerators by allowing bigger data sets to be indexed with limited memory.

The simple structure of our method allows it to be easily implemented using frameworks such as PyTorch, TensorFlow, and JAX. This enables `LoRANN` to support different hardware platforms with minimal differences between implementations. We write our GPU implementation in Python using JAX which uses XLA to compile and run the algorithm on a GPU or a TPU (Frostig et al., 2018).

5 Out-of-distribution queries

In the standard benchmark set-up (Li et al., 2019; Aumüller et al., 2020), a data set is randomly split into the corpus and a test set, i.e., the corpus and the queries are drawn from the same distribution. However, this assumption often does not hold in practice, for example in cross-modal search. Thus, there is recent interest (Simhadri et al., 2022; Jaiswal et al., 2022; Hyvönen et al., 2022) in the out-of-distribution (OOD) setting. For instance, in the Yandex-text-to-image data set, the corpus consists of images, and the queries are text; even though both the corpus and the queries are embedded in the same vector space, their distributions differ (see Figure 2 in Jaiswal et al., 2022).

Due to the regression formulation, the proposed method handles OOD queries by design. To verify this, we construct an index for a sample of 400K points of the Yandex OOD data set in four different scenarios: (1) the default version (`LoRANN-query`) uses $\{\mathbf{x}_i\}_{i=1}^n$, i.e., an $n = 400$K sample from the query distribution, as a global training set and selects the local training set $\{\mathbf{x}_i\}_{i \in J_l}$ as the global training set points that are routed into the lth cluster; (2) `LoRANN-query-big` is like `LoRANN-query`, but with $n = 1.2$M; (3) `LoRANN-corpus` uses the corpus $\{\mathbf{c}_j\}_{j=1}^m$ as a global training set, and selects the local training sets as $\{\mathbf{c}_j\}_{j \in J_l}$ like the default version; (4) `LoRANN-corpus-local` uses $\{\mathbf{c}_i\}_{j=1}^m$ as a global training set, but selects the local training sets as $\{\mathbf{c}_j\}_{j \in I_l}$, i.e., uses only the corpus points of the lth cluster to train the lth model. We can thus disentangle the effect of the choice of the global training set from the effect of using the points in the nearby clusters to train the RRR models.

The results are shown in Figure 1. The version trained on queries outperforms the version trained only on the corpus, especially in the case of no re-ranking. Furthermore, we can use larger training sets to increase the performance of `LoRANN`. Both `LoRANN-query` and `LoRANN-corpus` outperform `LoRANN-corpus-local`, indicating that selecting the local training sets as described in Section 3.2 improves the accuracy of the regression models. We assume that this is because of the larger and more representative training sets, even though they are not from the actual query distribution.

6 Related work

Supervised ANN algorithms Learning-to-hash methods (Weiss et al., 2008; Norouzi and Fleet, 2011; Liu et al., 2012) optimize partitions in a supervised fashion using data-dependent hash functions. Other supervised methods include learning optimal partitions by approximating a balanced graph partitioning (Dong et al., 2020; Gupta et al., 2022; Gottesbüren et al., 2024) and interpreting partitions as multilabel classifiers (Hyvönen et al., 2022). These supervised methods are orthogonal to our approach since they define the learning problem as selecting a subset of corpus points via partitioning. In contrast, we propose a supervised score computation method for clustering-based or, more generally, for partition-based ANN algorithms.

Product quantization The state-of-the-art clustering-based algorithms IVF-PQ (Jegou et al., 2011) and ScaNN (Guo et al., 2020) use quantization for data compression and score computation. They use a *quantizer* $q : \mathbb{R}^d \mapsto \mathcal{A}$ to map a point of the feature space to a value in a *codebook* $\mathcal{A}$. Given $\mathcal{A}$, they approximate the dissimilarity between the query point $\mathbf{x}$ and the corpus point $\mathbf{c}_j$ by $d(\mathbf{x}, q(\mathbf{c}_j))$, i.e., the dissimilarity between the query point and the codebook value corresponding

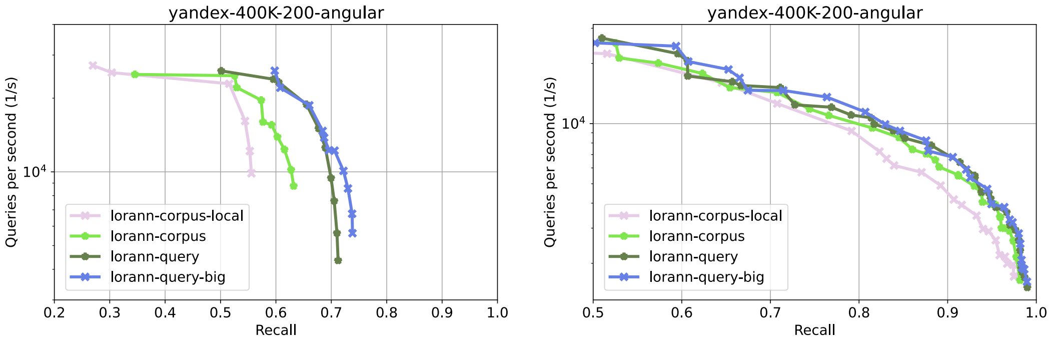


Figure 1: Recall vs. QPS on the Yandex T2I OOD data set (400K sampled corpus points) without (left) and with (right) the final re-ranking step. `LoRANN-query` is trained using a sample of 400K points from the query distribution as a training set, while `LoRANN-query-big` uses a sample of 1.2M points. `LoRANN-corpus` is trained using the corpus as a training set. `LoRANN-corpus-local` is trained using the corpus as a training set with only the cluster points as the local training sets of the reduced-rank regression models. It is beneficial to (1) use a sample from the actual query distribution as a training set and to (2) select the local training set by using also the points outside of the cluster as described in Section 3.2. The performance difference decreases when the final re-ranking step is introduced (requiring the original data set to be kept in memory).

to $\mathbf{c}_j$. Further, IVF-PQ and ScaNN quantize the residuals, i.e., the distances between corpus points and the cluster centroids, and use product quantization that decomposes the feature space into lower-dimensional subspaces and learns subquantizers in these subspaces. The code size, and thus the memory consumption, of PQ is directly proportional to the number of subquantizers.

7 Experiments

We use the ANN-benchmarks project (Aumüller et al., 2020) to run our experiments[3] and replicate its experimental set-up as closely as possible (see Appendix B for the description of the experimental set-up). We use $k = 100$ for all experiments and measure recall (the proportion of true k-nn found) versus queries per second (QPS). Additionally, due to the lack of modern high-dimensional embedding data sets in ANN-benchmarks, we include multiple new high-dimensional embedding data sets in our experiments; for a description of all the data sets, see Appendix C.

Note that, even though we demonstrated in Section 5 that `LoRANN` can adapt to the query distribution, there are no samples from the actual query distribution available for the benchmark data sets of this section. Thus, we follow the standard approach by using only the corpus $\{\mathbf{c}_j\}_{j=1}^m$ to train `LoRANN`.

7.1 Reduced-rank regression

We first compare the proposed score computation method, reduced-rank regression (RRR), against product quantization (PQ) for clustering-based ANN search.[4] We use RRR and PQ as scoring methods for an IVF index that partitions the corpus using k-means (see Section 2.2). We implement RRR using 8-bit integer quantization as described in Section 3.3 and compare against product quantization implemented in Faiss (Douze et al., 2024) with 4-bit integer quantization and fast scan (André et al., 2015). First, we compare the score computation methods at the optimal hyperparameters while keeping the clustering fixed, and then compare them at a fixed memory budget.

[3] `https://github.com/ejaasaari/lorann-experiments`

[4] Since ScaNN does not outperform Faiss-IVFPQ in our end-to-end-evaluation (Section 7.2.2) and both methods employ k-means for partitioning, the anisotropic quantization (Guo et al., 2020) used by ScaNN is unlikely to outperform the original PQ (Jegou et al., 2011).

7.1.1 Fixed clustering

To directly compare the score computation methods, in Figure 2 we present results where the IVF index (the partition defined by k-means clustering) is the same for both methods. For RRR, we use $r = 32$, while for PQ each vector is encoded with $d/2$ subquantizers for optimal performance. RRR outperforms PQ on seven out of the eight data sets; for complete results, see Appendix D.1.

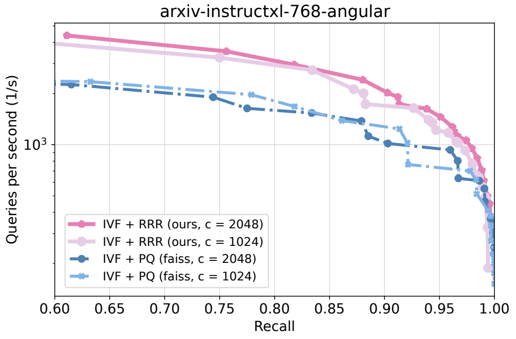


Figure 2: Performance comparison between RRR and PQ. The k-means clustering (IVF index) is kept constant to directly compare the effect of the score computation method (here c denotes the number of clusters). The proposed score computation method outperforms the baseline method (PQ).

7.1.2 Memory usage

In Figure 3, we compare the performance of RRR (IVF+RRR) and PQ (IVF+PQ) by varying the rank parameter r for RRR and the code size for PQ such that b, bytes per vector, is similar for both. For all values of b, RRR outperforms PQ which is a typical choice in memory-limited use cases. Note that RRR with $b \approx 16$ outperforms PQ even with $b \approx 64$ on all data sets; for the full results, see Appendix D.2. The results are similar when no final re-ranking step is used (Appendix D.3).

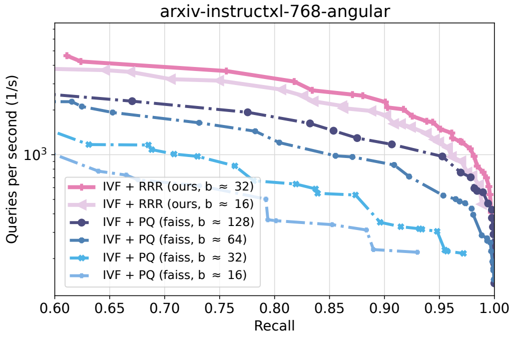


Figure 3: Performance comparison of RRR and PQ at different levels of memory usage. We vary the rank parameter r for RRR and the code size for PQ such that b, bytes per vector, is similar for both. RRR@$(b \approx 16)$ outperforms even PQ@$(b \approx 128)$ which uses eight times as much memory.

7.2 LoRANN

In this section, we measure the end-to-end performance of `LoRANN` (see Section 4). We first perform an ablation study on the components of `LoRANN` (Section 7.2.1), and then perform an end-to-end evaluation of `LoRANN` against the state-of-the-art ANN libraries in both the CPU setting (Section 7.2.2) and the GPU setting (Section 7.2.3).

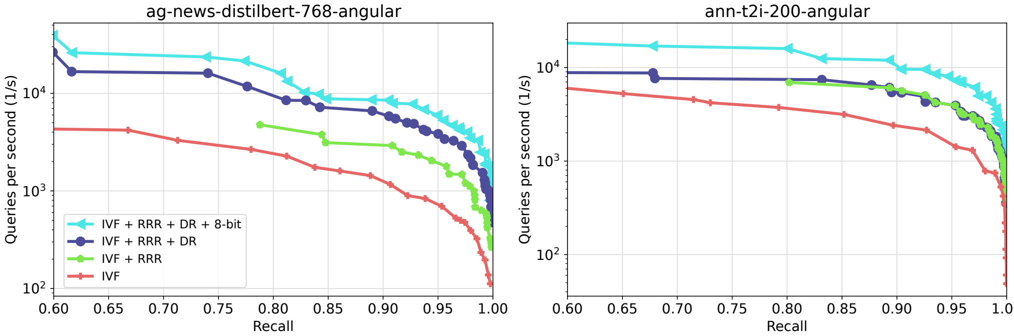


Figure 4: LoRANN ablation study. On the high-dimensional ($d = 768$) data set (left), all the components improve the performance of LoRANN. On the lower-dimensional ($d = 200$) data set (right), all the components except dimensionality reduction (DR) improve performance.

7.2.1 Ablation study

In Figure 4, we study the effect of the different components of LoRANN on its performance (for the full results, see Appendix E.1). As the baseline, we use an IVF index. Adding the score computation step via RRR significantly improves performance on all the data sets.

Dimensionality reduction (DR) is beneficial on higher-dimensional data sets with moderate-sized corpora: if the number of points in a cluster is lower than the dimension, then the first vector-matrix product in the computation $(\mathbf{x}^T\mathbf{A})\mathbf{B}$ of the local reduced-rank regression models will be more expensive. For large data sets with high-dimensional data, this effect decreases. Dimensionality reduction also does not improve performance on the lower-dimensional data sets ($d \leq 300$). Incorporating 8-bit quantization improves not only the memory usage but also the query latency.

In Appendix E.2, we study the effect of varying the rank r of the parameter matrices. We find that increasing r from 32 to 64 has little effect, while $r = 16$ performs worse for high-dimensional data (but can be used to further decrease memory usage). For all of our other experiments, we use $r = 32$.

7.2.2 CPU evaluation

As the baseline methods, we choose four leading graph implementations, HNSW, GLASS[5], QSG-NGT, and PyNNDescent (Dong et al., 2011), two leading product quantization implementations, Faiss-IVFPQ (fast scan) and ScaNN, and the leading tree implementation MRPT (Hyvönen et al., 2016). See Figure 5 for the results and Appendix E.3 for the results on all 16 data sets. Three trends emerge: (1) LoRANN outperforms the product quantization methods on all data sets except glove-200-angular. (2) LoRANN performs better in the high-dimensional regime: it outperforms all the other methods except GLASS and QSG-NGT on all but the lower-dimensional ($d \leq 200$) data sets. (3) Compared to graph methods, LoRANN performs better at the lower recall levels: QPS-recall curves of LoRANN and GLASS cross between 80% and 99% on most of the data sets. On 8 of the 16 data sets, LoRANN has better or similar performance as QSG-NGT at all recall levels.

Furthermore, in Appendix E.4, we demonstrate that in general LoRANN has faster index construction times than the graph-based methods.

7.2.3 GPU evaluation

We compare LoRANN against GPU implementations of IVF and IVF-PQ in both Faiss (Douze et al., 2024) and the NVIDIA RAFT library[6]. In addition, we compare against a state-of-the-art GPU graph algorithm CAGRA (Ootomo et al., 2023) implemented in RAFT. All algorithms receive all test queries as one batch of size 1000. See Figure 6 for representative results, and Appendix E.5 for the complete results. LoRANN outperforms the other methods on seven out of the nine high-dimensional ($d > 300$) data sets. For $d \leq 300$, CAGRA has the best performance.

[5] An efficient HNSW implementation with quantization: https://github.com/zilliztech/pyglass

[6] https://github.com/rapidsai/raft

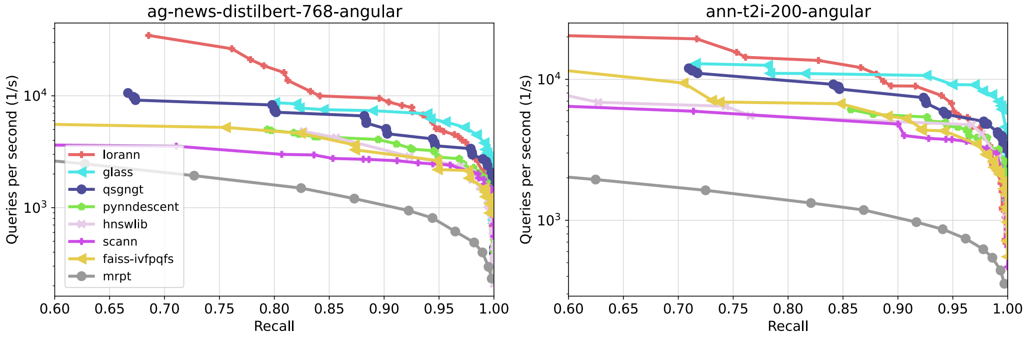


Figure 5: CPU comparison. The QPS-recall curves of `LoRANN` and the leading graph library GLASS cross at the 95% (left) and at the 90% recall level (right), indicating that `LoRANN` is the fastest method at the lower recall levels, and GLASS at the higher recall levels.

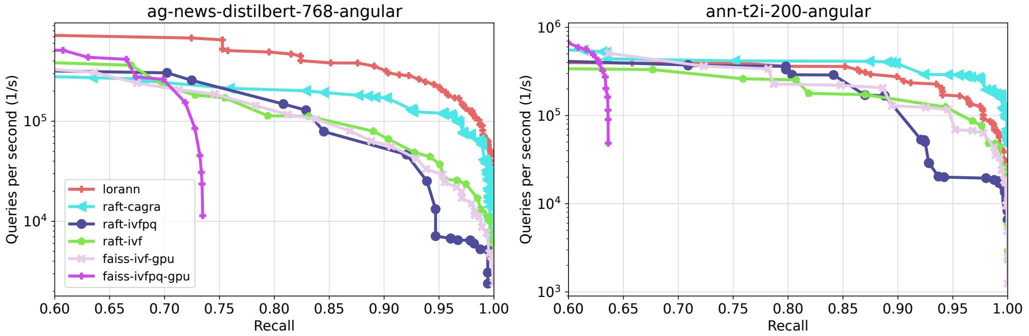


Figure 6: GPU comparison. On the high-dimensional $(d = 768)$ data set (left), `LoRANN` is the fastest, and on the lower-dimensional $(d = 200)$ data set (right), the graph method CAGRA is the fastest.

To demonstrate the ease of implementation of our algorithm for new hardware platforms, in Appendix E.5 we implement our method for Apple silicon. We show using the M2 Pro SoC that `LoRANN` can take advantage of the M2 GPU and its unified memory architecture to achieve faster queries.

8 Discussion

In this article, we show that an elementary statistical method, reduced-rank regression (RRR), is surprisingly efficient for score computation in clustering-based ANN search. Since RRR outperforms product quantization (PQ) at fixed memory consumption while being simpler to implement efficiently, we recommend using it in regimes where PQ is traditionally used, e.g., when the memory usage is a limiting factor (Douze et al., 2024). While the experiments of the article are performed in the standard in-memory setting, the simple structure and the small memory footprint of the proposed method suggest scalability for larger data sets that do not fit into the main memory. In particular, hybrid solutions that store the corpus on an SSD (Jayaram Subramanya et al., 2019; Ren et al., 2020; Chen et al., 2021) and the distributed setting where the corpus and the index are distributed over multiple machines (Deng et al., 2019; Gottesbüren et al., 2024) are promising research directions.

Limitations Since reaching the highest recall levels ($> 90\%$ for $k = 100$) requires exploring many clusters, graph methods are usually more efficient than clustering-based methods in this regime. Furthermore, low-dimensional data sets (e.g., $d < 100$) also require that the rank r of the parameter matrix is reasonably high. Thus, reduced-rank regression is not efficient for low-dimensional data sets for which the proportion r/d is too large. The proposed score computation method is also only applicable to inner product-based dissimilarity measures. However, the multivariate regression formulation of Section 3.1 can be extended for other dissimilarity measures.

Acknowledgments

This work has been supported by the Research Council of Finland (grant #345635 and the Flagship programme: Finnish Center for Artificial Intelligence FCAI) and the Jane and Aatos Erkko Foundation (BioDesign project, grant #7001702). We acknowledge the computational resources provided by the Aalto Science-IT Project from Computer Science IT. We thank the anonymous reviewers for their valuable feedback.

References

Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan. *Proceedings of the VLDB Endowment*, 9(4):288–299, 2015.

Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *Proceedings of the International Conference on Machine Learning*, pages 2206–2240. PMLR, 2022.

Sebastian Bruch. *Foundations of Vector Retrieval*. Springer, 2024.

Sebastian Bruch, Aditya Krishnan, and Franco Maria Nardini. Optimistic query routing in clustering-based approximate maximum inner product search. *arXiv preprint arXiv:2405.12207*, 2024.

Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002.

Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. SPANN: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems*, 34:5199–5212, 2021.

Sanjoy Dasgupta and Kaushik Sinha. Randomized partition trees for nearest neighbor search. *Algorithmica*, 72(1):237–263, 2015.

Rieke de Maeyer, Sami Sieranoja, and Pasi Fränti. Balanced k-means revisited. *Applied Computing and Intelligence*, 3(2):145–179, 2023.

Shiyuan Deng, Xiao Yan, KW Ng Kelvin, Chenyu Jiang, and James Cheng. Pyramid: A general framework for distributed similarity search on large-scale datasets. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1066–1071. IEEE, 2019.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35: 30318–30332, 2022.

Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine learning*, 42:143–175, 2001.

Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, pages 577–586, 2011.

Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *Proceedings of the International Conference on Learning Representations*, 2020.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The Faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. *Machine Learning and Systems (MLSys)*, 2018.

Lars Gottesbüren, Laxman Dhulipala, Rajesh Jayaram, and Jakub Lacki. Unleashing graph partitioning for large-scale nearest neighbor search. *arXiv preprint arXiv:2403.01797*, 2024.

Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P. A. Lensch. GGNN: Graph-based gpu nearest neighbor search. *IEEE Transactions on Big Data*, 9(1):267–279, 2022.

Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, et al. Manu: a cloud native vector database management system. *Proceedings of the VLDB Endowment*, 15(12):3548–3561, 2022.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *Proceedings of the International Conference on Machine Learning*, pages 3887–3896. PMLR, 2020.

Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J Smola. BLISS: A billion scale index using iterative re-partitioning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 486–495, 2022.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *Proceedings of the International Conference on Machine Learning*, pages 3929–3938. PMLR, 2020.

Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2): 217–288, 2011.

Ville Hyvönen, Teemu Pitkänen, Sotiris Tasoulis, Elias Jääsaari, Risto Tuomainen, Liang Wang, Jukka Corander, and Teemu Roos. Fast nearest neighbor search through sparse random projections and voting. In *Proceedings of the 2016 IEEE International Conference on Big Data*, pages 881–888. IEEE, 2016.

Ville Hyvönen, Elias Jääsaari, and Teemu Roos. A multilabel classification framework for approximate nearest neighbor search. *Advances in Neural Information Processing Systems*, 35: 35741–35754, 2022.

Masajiro Iwasaki and Daisuke Miyazaki. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *arXiv preprint arXiv:1810.07355*, 2018.

Alan Julian Izenman. Reduced-rank regression for the multivariate linear model. *Journal of multivariate analysis*, 5(2):248–264, 1975.

Elias Jääsaari, Ville Hyvönen, and Teemu Roos. Efficient autotuning of hyperparameters in approximate nearest neighbor search. In *Proceedings of the 23rd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, volume 2, pages 590–602. Springer, 2019.

Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. OOD-DiskANN: Efficient and scalable graph ANNS for out-of-distribution queries. *arXiv preprint arXiv:2211.12850*, 2022.

Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems*, 32, 2019.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *Proceedings of the International Conference on Learning Representations*, 2020.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2019.

Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *Proceedings of the International Conference on Machine Learning*, pages 1–8. PMLR, 2011.

Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081. IEEE, 2012.

Zepu Lu, Jin Chen, Defu Lian, Zaixi Zhang, Yong Ge, and Enhong Chen. Knowledge distillation for high dimensional search index. In *Advances in Neural Information Processing Systems*, volume 36, pages 33403–33419, 2023.

Yu A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2018.

Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

Mohammad Norouzi and David J. Fleet. Minimal loss hashing for compact binary codes. In *Proceedings of the International Conference on Machine Learning*, pages 353–360. PMLR, 2011.

Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. CAGRA: Highly parallel graph construction and approximate nearest neighbor search for GPUs. *arXiv preprint arXiv:2308.15136*, 2023.

James Jie Pan, Jianguo Wang, and Guoliang Li. Vector database management techniques and systems. In *Companion of the 2024 International Conference on Management of Data*, pages 597–604, 2024.

Jie Ren, Minjia Zhang, and Dong Li. HM-ANN: Efficient billion-point nearest neighbor search on heterogeneous memory. *Advances in Neural Information Processing Systems*, 33:10672–10684, 2020.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 8371–8384. Association for Computational Linguistics, 2024.

Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. Results of the NeurIPS'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 177–189. PMLR, 2022.

Philip Sun, David Simcha, Dave Dopson, Ruiqi Guo, and Sanjiv Kumar. SOAR: Improved indexing for approximate nearest neighbor search. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Thomas Vecchiato, Claudio Lucchese, Franco Maria Nardini, and Sebastian Bruch. A learning-to-rank formulation of clustering-based approximate nearest neighbor search. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2261–2265, 2024.

Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. *Advances in Neural Information Processing Systems*, 33:21665–21674, 2020.

Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2614–2627, 2021.

Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. *Advances in Neural Information Processing Systems*, 21:1753–1760, 2008.

Weijie Zhao, Shulong Tan, and Ping Li. SONG: Approximate nearest neighbor search on GPU. In *IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1033–1044. IEEE, 2020.

Xi Zhao, Bolong Zheng, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. FARGO: Fast maximum inner product search via global multi-probing. *Proceedings of the VLDB Endowment*, 16(5):1100–1112, 2023.

A Geometric intuition

Observe that the reduced-rank regression solution is different from the most obvious low-rank approximation of the OLS solution computed via an SVD of the matrix $\mathbf{C}$. In Section 5, we verify experimentally using a real-world data set that a more accurate model can indeed be trained by factorizing $\mathbf{X}\mathbf{C}^T$ instead of $\mathbf{C}$. These two solutions are equivalent only in the special case where $\mathbf{X} = \mathbf{C}$. In this special case, computing the predictions of the reduced-rank regression model, denoted by $\hat{\mathbf{y}}_{\text{RRR}} := \mathbf{x}^T \hat{\boldsymbol{\beta}}_{\text{RRR}}$, is equivalent to projecting both the query point $\mathbf{x}$ and the cluster points $\{\mathbf{c}_j\}_{j \in I_j}$ onto the subspace spanned by the first r eigenvectors of $\mathbf{K} := \mathbf{C}^T\mathbf{C}$, and computing the inner products in this r-dimensional subspace. If the cluster points are centered, the eigenvectors of $\mathbf{K}$ are the principal axes of the cluster points, and $\frac{1}{m_l - 1}\mathbf{K}$ can be interpreted as the sample covariance matrix. However, unlike in principal component analysis (PCA), data should not be centered when estimating the inner products, since the inner products are not invariant with respect to translation.

Let $\mathbf{W}_r \in \mathbb{R}^{d \times r}$ be a matrix containing the first r eigenvectors of $\mathbf{K}$ as columns, and denote by $\tilde{\mathbf{x}} := \mathbf{W}_r^T \mathbf{x}$ the r-dimensional projection of the query point, by $\tilde{\mathbf{C}} := \mathbf{C}\mathbf{W}_r$ an $m_l \times r$-matrix containing the projected cluster points, and by $\tilde{\mathbf{y}} := \tilde{\mathbf{x}}^T \tilde{\mathbf{C}}^T$ an $1 \times m_l$-matrix containing the inner products between the projected query and the projected cluster points. Using this notation, we have the following result:

Theorem 1. *If $\mathbf{X} = \mathbf{C}$, then $\tilde{\mathbf{y}} = \hat{\mathbf{y}}_{RRR}$.*

Proof. Denote $\tilde{\boldsymbol{\beta}} := \mathbf{W}_r\mathbf{W}_r^T\mathbf{C}$. Now $\tilde{\mathbf{y}}$ can be written as $\tilde{\mathbf{y}} = \mathbf{x}^T\tilde{\boldsymbol{\beta}}$. To prove the result, it suffices to show that $\tilde{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}_{\text{RRR}}$. Denote the singular value decomposition of $\mathbf{C}^T$ by $\mathbf{C}^T = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$. Since $\mathbf{X} = \mathbf{C}$, $\mathbf{Y} = \mathbf{C}\mathbf{C}^T$, and, consequently, the right singular vectors of $\mathbf{C}^T$ are the eigenvectors of $\mathbf{Y}$. Thus, the reduced-rank regression solution is

$$\hat{\boldsymbol{\beta}}_{\text{RRR}} = \mathbf{C}^T\mathbf{V}_r\mathbf{V}_r^T,$$

where $\mathbf{V}_r$ is $m \times r$-matrix containing the first r columns of $\mathbf{V}$.

The columns of the matrix $\mathbf{U}$—i.e., the left singular vectors of $\mathbf{C}^T$—are the eigenvectors of $\mathbf{K} = \mathbf{C}^T\mathbf{C}$. Thus, $\mathbf{W}_r = \mathbf{U}_r$, and $\tilde{\boldsymbol{\beta}}$ can be written as

$$\tilde{\boldsymbol{\beta}} = \mathbf{U}_r\mathbf{U}_r^T\mathbf{C},$$

where we denote by $\mathbf{U}_r$ a $d \times r$-matrix containing the first r columns of the matrix $\mathbf{U}$. Since $\mathbf{U}^T\mathbf{C}^T = \boldsymbol{\Sigma}\mathbf{V}^T$, also $\mathbf{U}_r^T\mathbf{C}^T = \boldsymbol{\Sigma}_r\mathbf{V}_r^T$, and since $\mathbf{C}^T\mathbf{V} = \mathbf{U}\boldsymbol{\Sigma}$, also $\mathbf{C}^T\mathbf{V}_r = \mathbf{U}_r\boldsymbol{\Sigma}_r$. Using these identities, we have

$$\tilde{\boldsymbol{\beta}} = \mathbf{U}_r\mathbf{U}_r^T\mathbf{C}^T = \mathbf{U}_r\boldsymbol{\Sigma}_r\mathbf{V}_r^T = \mathbf{C}^T\mathbf{V}_r\mathbf{V}_r^T = \hat{\boldsymbol{\beta}}_{\text{RRR}},$$

which completes the proof. □

However, in the general case where $\mathbf{X} \neq \mathbf{C}$, i.e., the training set of the lth model is selected as the training queries that are routed into the lth cluster (for which $l \in \tau(\mathbf{x}_i)$), the matrix $\mathbf{X}$ also affects the right singular vectors of the output matrix $\mathbf{Y} = \mathbf{X}\mathbf{C}^T$. Hence, there is no simple geometric interpretation for the reduced-rank regression solution $\hat{\boldsymbol{\beta}}_{\text{RRR}}$ in the d-dimensional feature space.

B Experimental set-up

We use the ANN-benchmarks project[7] (Aumüller et al., 2020) to run our experiments and replicate its set-up as close as possible: our experiments are performed on AWS `r6i.4xlarge` instances with Intel Xeon 8375C (Ice Lake) processors and hyperthreading disabled, and all algorithms are run using a single core. For our GPU experiments, we use an AWS `g5.2xlarge` instance with an NVIDIA A10G GPU (24 GB VRAM) and a `mac2-m2pro.metal` instance with an Apple M2 Pro SoC.

We use $k = 100$ for all experiments and measure the recall (the proportion of true k-nn found) versus queries per second (QPS). For all data sets, we use a separate test set of 1000 queries, and each hyperparameter combination is benchmarked 5 times, from which the lowest achieved query latency is recorded. Additionally, due to the lack of modern high-dimensional embedding data sets in ANN-benchmarks, we include multiple new high-dimensional data sets in our experiments; for a list of all the data sets, see Appendix C. To account for the new data sets, we increase the sizes of the considered hyperparameter grids in applicable cases.

For `LoRANN` experiments (Section 7.2), we use the following hyperparameter grid:

- Number of clusters L: 1024, 2048, 4096
- Reduced dimension s: 64, 128, 192
- Rank r: 32 (for GPU, r: 24)
- Clusters to search w: 8, 16, 24, 32, 48, 64, 128, 256
- Points to re-rank t: 100, 200, 400, 800, 1200, 1600, 2400, 3200

For experiments on just reduced-rank regression (Section 7.1), we use the same hyperparameter grid for L, w, and t as above and do not use dimensionality reduction.

For all hyperparameters, see `https://github.com/ejaasaari/lorann-experiments`.

[7] `https://github.com/erikbern/ann-benchmarks`

C Data sets

In addition to data sets from ANN-benchmarks (fashion-mnist, gist, glove, mnist, sift), we include high-dimensional neural network embedding data sets that are more representative of those encountered in modern machine learning applications and encompass dimensionalities ranging from 200 to 1536. The full list of data sets along with their sizes and the distance metrics used is given in Table 1.

Table 1: Data sets used in the experiments.

Data set (model)	type	corpus size	dim	distance	license
ag-news (DistilBERT)[1]	text embedding	120 000	768	angular	CC BY 4.0
ag-news (MiniLM)[1]	text embedding	120 000	384	angular	CC BY 4.0
ann-arxiv (E5-base)[2]	text embedding	2 288 300	768	angular	Apache 2.0
ann-t2i (ResNext)[3]	image embedding	1 000 000	200	angular	Apache 2.0
arxiv (OpenAI Ada)[4]	text embedding	319 224	1536	angular	CC0 1.0
arxiv (InstructXL)[5]	text embedding	2 254 000	768	angular	N/A
fashion-mnist[6]	raw image	60 000	784	euclidean	MIT
fasttext-wiki[7]	word embedding	1 000 000	300	euclidean	CC BY-SA 3.0
gist[8]	image descriptor	1 000 000	960	euclidean	CC0 1.0
glove[9]	word embedding	1 193 514	200	angular	Apache 2.0
mnist[10]	raw image	60 000	784	euclidean	CC BY-SA 3.0
sift[8]	image descriptor	1 000 000	128	euclidean	CC0 1.0
wiki (GTE-small)[11]	text embedding	224 482	384	angular	MIT
wiki (OpenAI Ada)[11]	text embedding	224 482	1536	angular	MIT
wolt (clip-ViT)[12]	image embedding	1 720 611	512	angular	N/A
yandex T2I 5M[13]	text/image	5 000 000	200	angular	CC BY 4.0

[1] https://data.dtu.dk/articles/dataset/Pretrained_sentence_BERT_models_AG_News_embeddings/21286923

[2] https://huggingface.co/datasets/unum-cloud/ann-arxiv-2m

[3] https://huggingface.co/datasets/unum-cloud/ann-t2i-1m

[4] https://www.kaggle.com/datasets/awester/arxiv-embeddings

[5] https://huggingface.co/datasets/Qdrant/arxiv-abstracts-instructorxl-embeddings

[6] https://github.com/zalandoresearch/fashion-mnist

[7] https://fasttext.cc/docs/en/english-vectors.html

[8] http://corpus-texmex.irisa.fr/

[9] https://nlp.stanford.edu/projects/glove/

[10] https://yann.lecun.com/exdb/mnist/

[11] https://huggingface.co/datasets/Supabase/wikipedia-en-embeddings

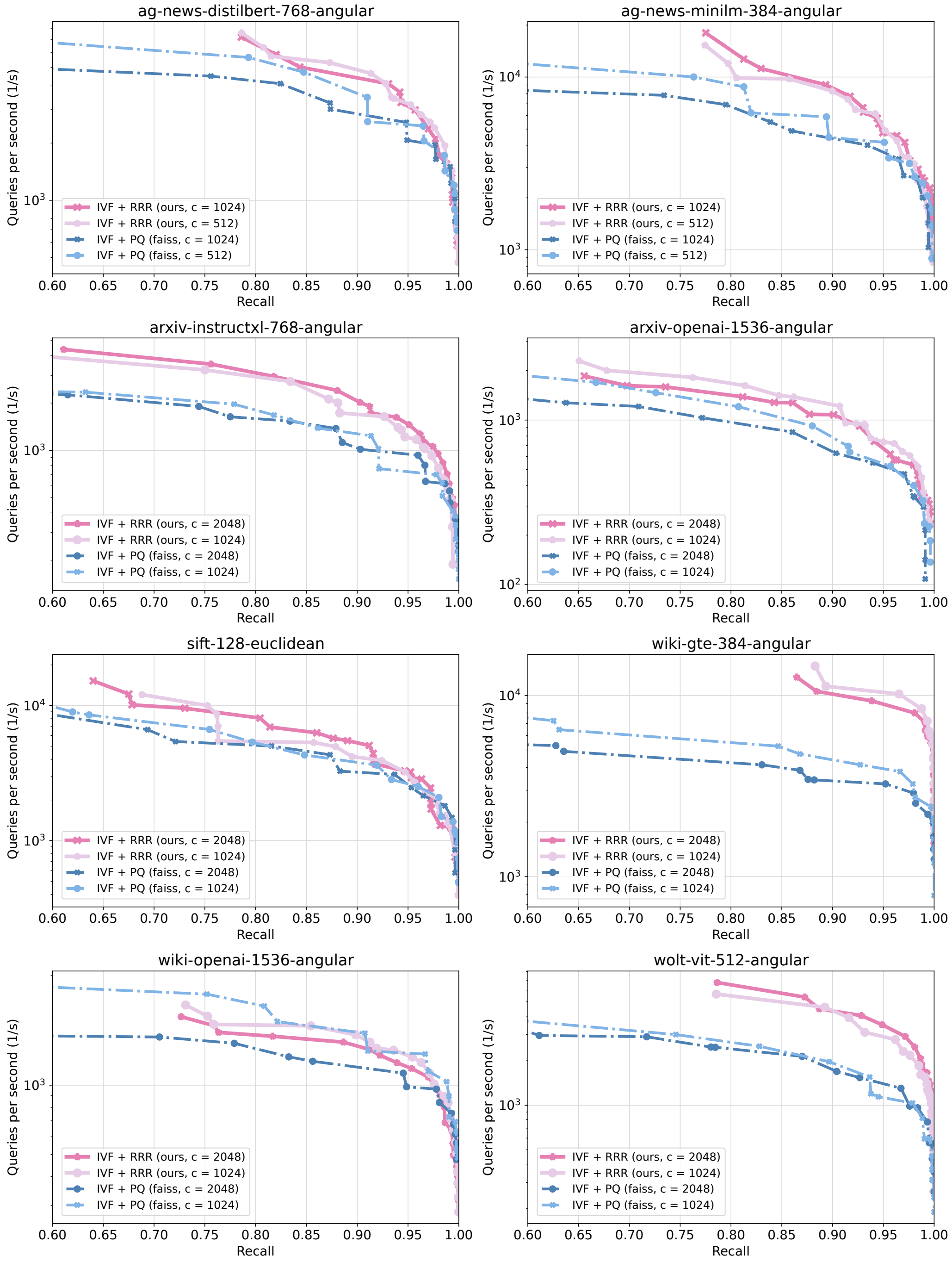
[12] https://huggingface.co/datasets/Qdrant/wolt-food-clip-ViT-B-32-embeddings

[13] https://big-ann-benchmarks.com/neurips23.html

D Reduced-rank regression experiments

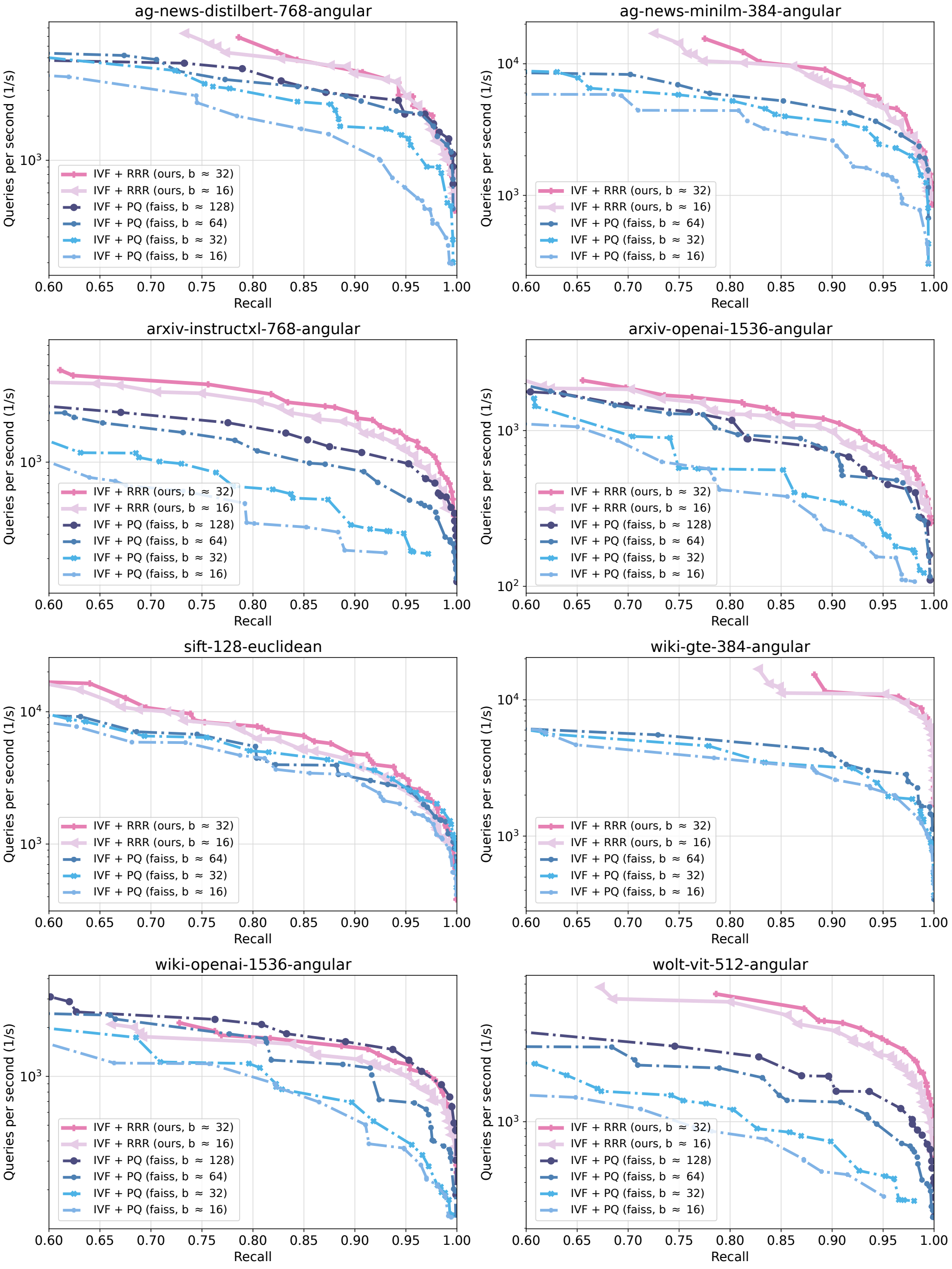
D.1 Fixed clustering

In this section, we present the complete evaluation of reduced-rank regression in comparison to product quantization when the k-means clustering is kept fixed. For discussion, refer to Section 7.1.1.



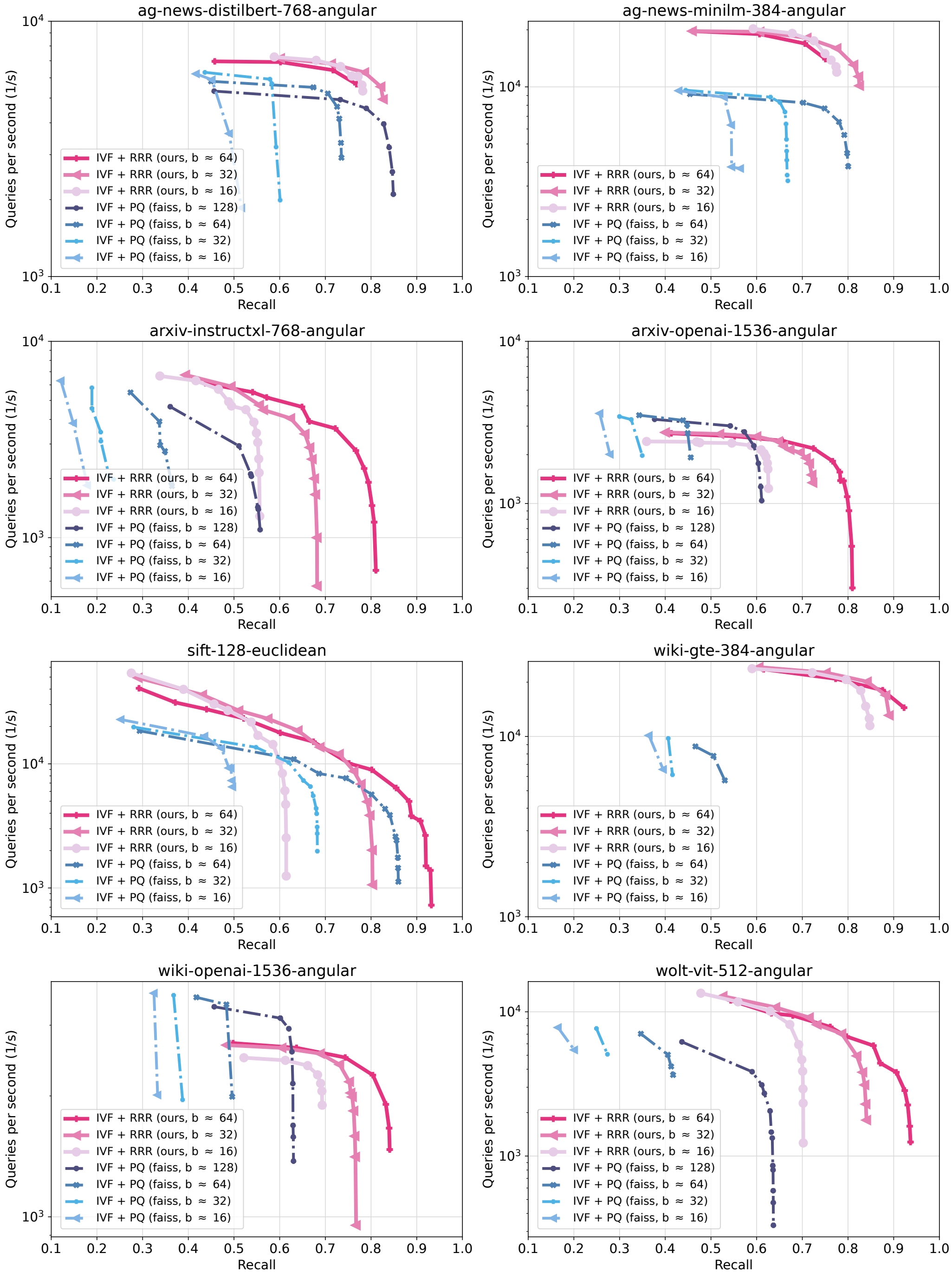
D.2 Memory usage

In this section, we present the complete evaluation of reduced-rank regression in comparison to product quantization for different memory consumptions. For discussion, refer to Section 7.1.2.



D.3 Memory usage (no re-ranking)

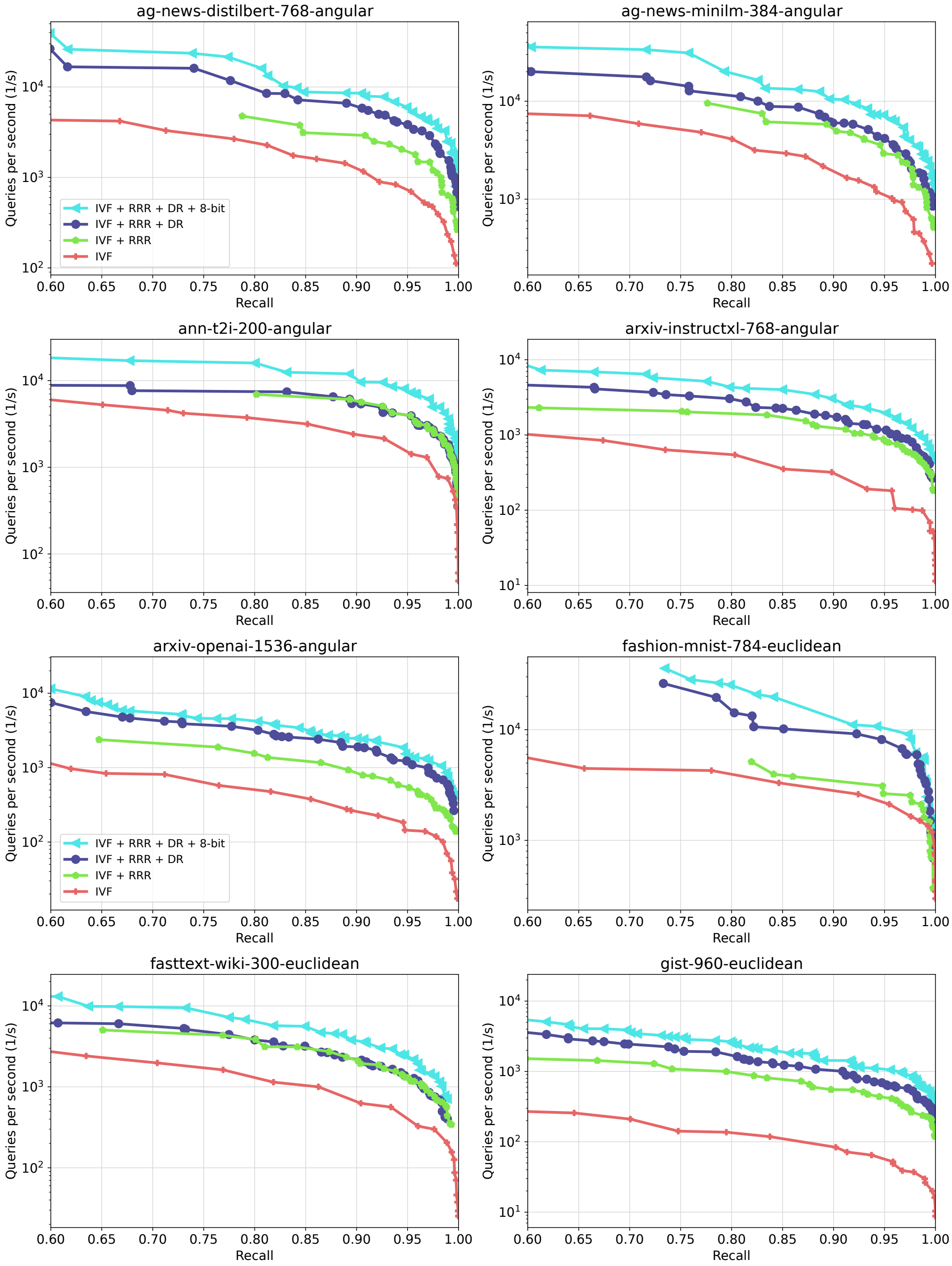
In this section, we present the evaluation of reduced-rank regression in comparison to product quantization for different memory consumptions when no final re-ranking step is used. In this scenario, the original data set does not have to be kept in memory.

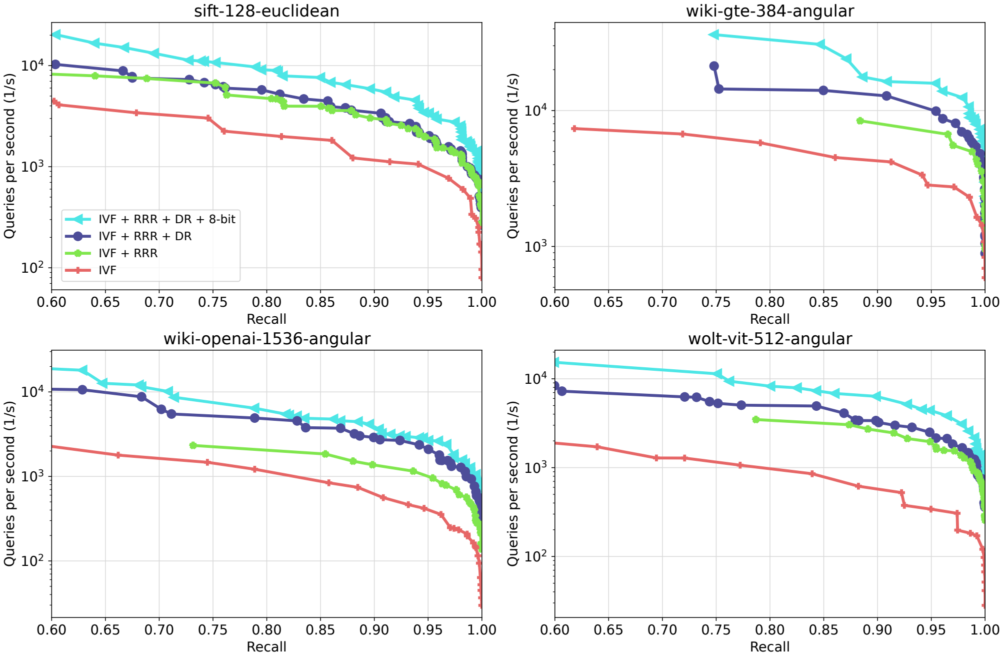


E LoRANN experiments

E.1 Ablation study

In this section, we study the effect of the components (reduced-rank regression, dimensionality reduction, 8-bit quantization) of LoRANN on its performance. For discussion, refer to Section 7.2.1.





E.2 Effect of the rank

In Figure 7, we study the impact of the rank r of the parameter matrices $\hat{\beta}_{\text{RRR}}$. We find that LoRANN is robust with respect to the choice of r: increasing r from 32 to 64 has little effect (when the final re-ranking step is used), while $r = 16$ performs only slightly worse for high-dimensional data. In our experiments, we use $r = 32$, but $r = 16$ can be used to decrease the memory consumption, and $r = 64$ can be used to achieve higher recall if the final re-ranking step is not used.

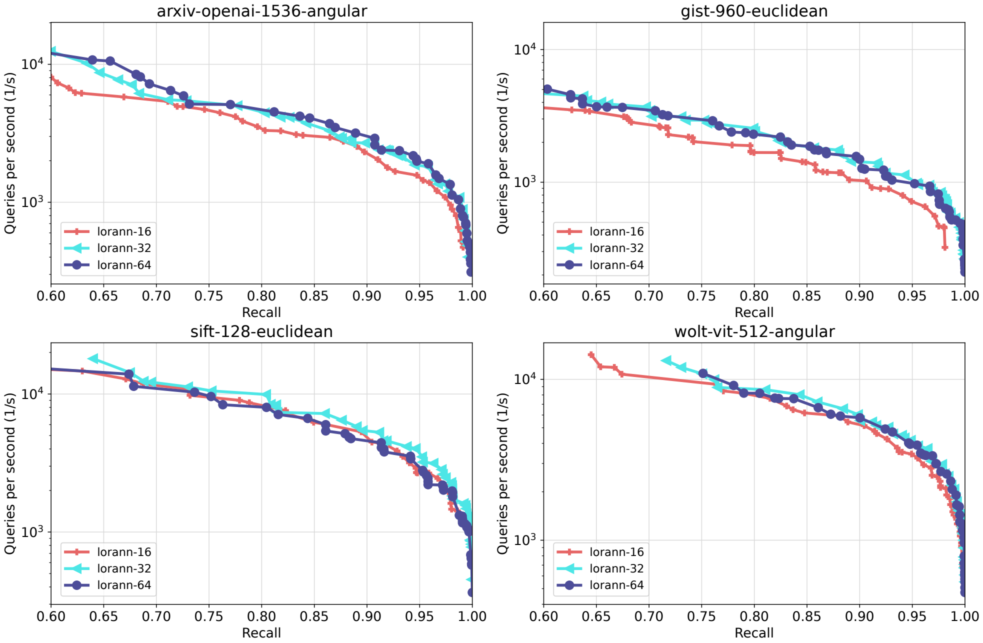
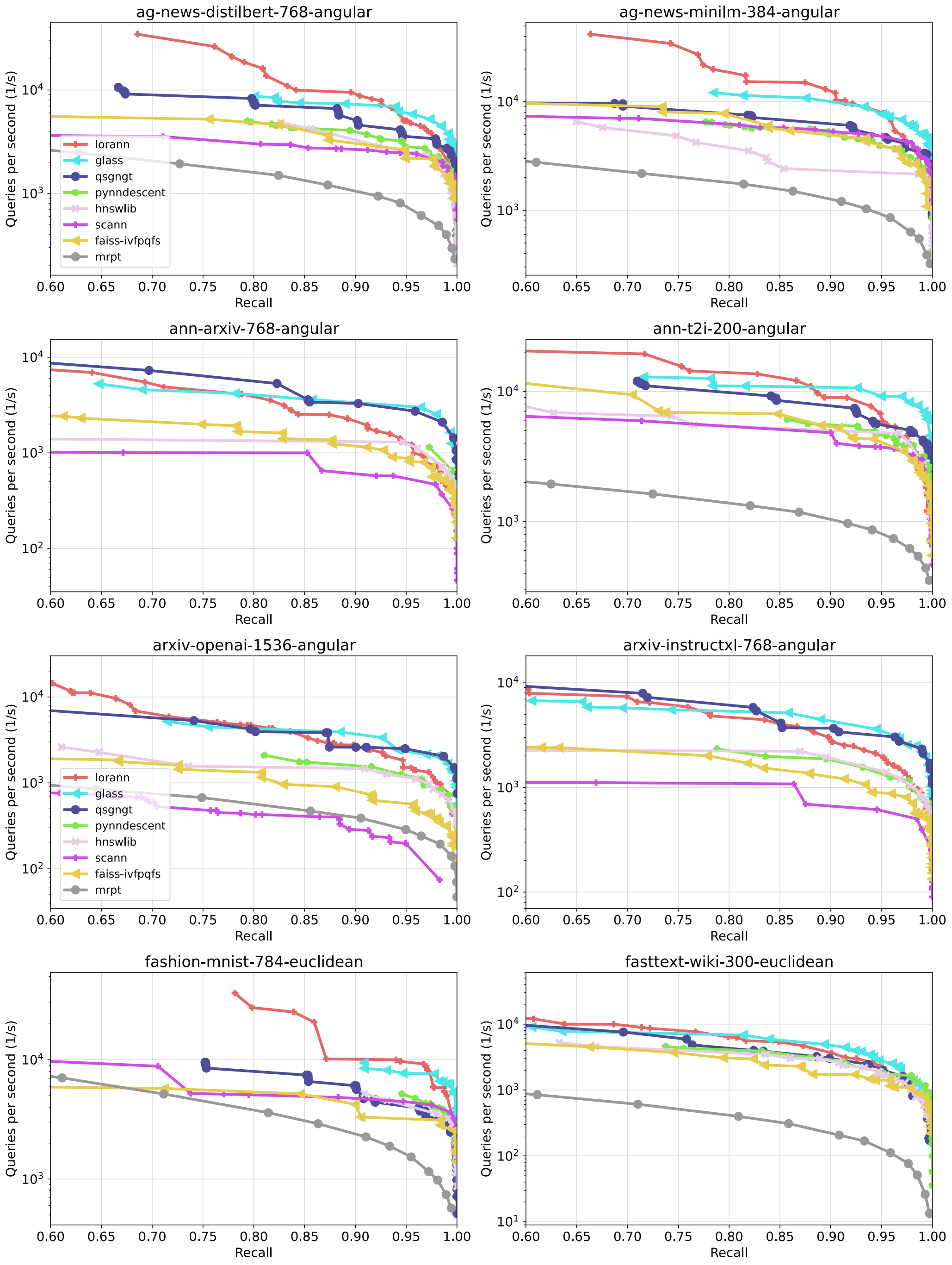
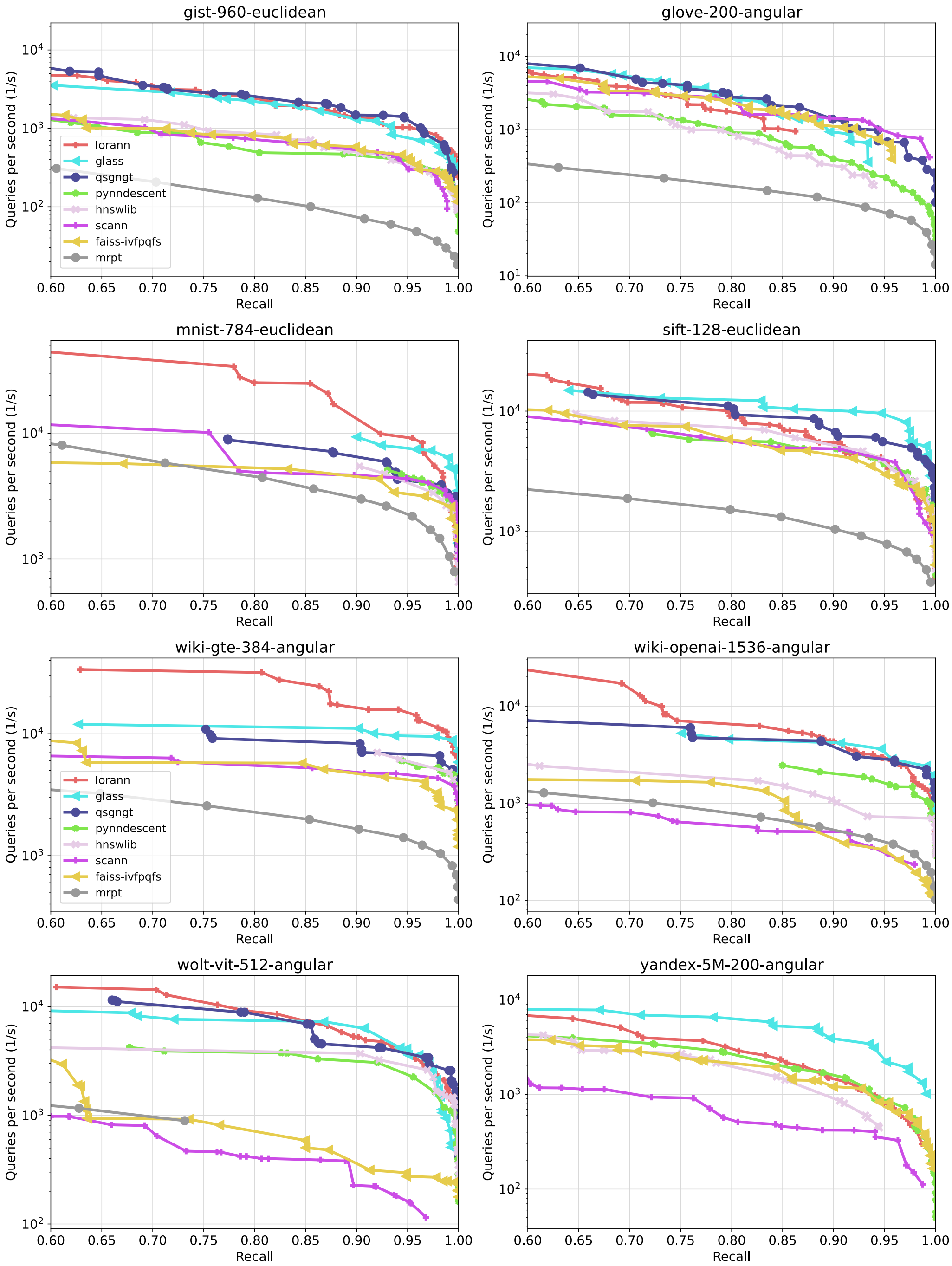


Figure 7: Effect of the rank r of the parameter matrices $\hat{\beta}_{\text{RRR}}$ in LoRANN.

E.3 CPU Evaluation

In this section, we present the complete evaluation of LoRANN in the CPU setting in comparison to other ANN algorithms. For discussion, refer to Section 7.2.2.




gist-960-euclidean
glove-200-angular
mnist-784-euclidean
sift-128-euclidean
wiki-gte-384-angular
wiki-openai-1536-angular
wolt-vit-512-angular
yandex-5M-200-angular
Queries per second (1/s)
Recall
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
10^1
10^2
10^3
10^4
lorann
glass
qsgngt
pynndescent
hnswlib
scann
faiss-ivfpqfs
mrpt

E.4 Index construction time

For all libraries, we measure the index construction time on a single CPU core (for LoRANN, the index construction parallelizes easily as all local cluster models can be computed independently). Table 2 shows index construction times for all data sets and compared methods at 90% recall for $k = 100$ at the optimal hyperparameters (a dash means that the algorithm either did not reach the necessary recall or ran out of memory on the data set).

LoRANN has slower index construction times than IVFPQfs, and similar index construction times as ScaNN. In general, LoRANN has faster index construction times than the graph methods: QSG-NGT has extremely slow index construction times, HNSW has slower index construction times than LoRANN on all data sets except ann-t2i, and GLASS is slower on all data sets except two lower-dimensional data sets (ann-t2i and fasttext-wiki).

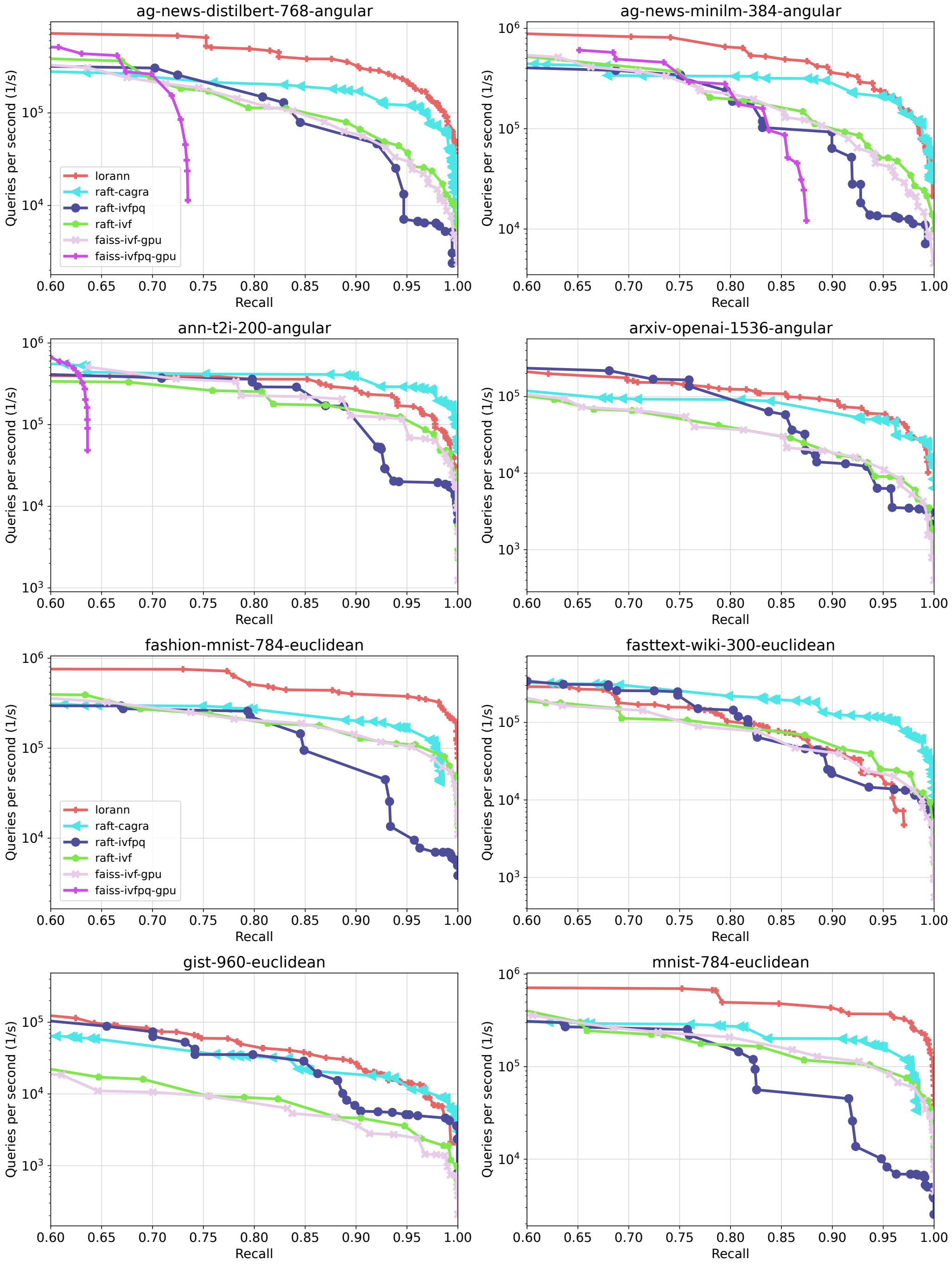
We note that our implementation of LoRANN has not been optimized with respect to index construction time, and could be improved with further sampling and more approximate SVD computations.

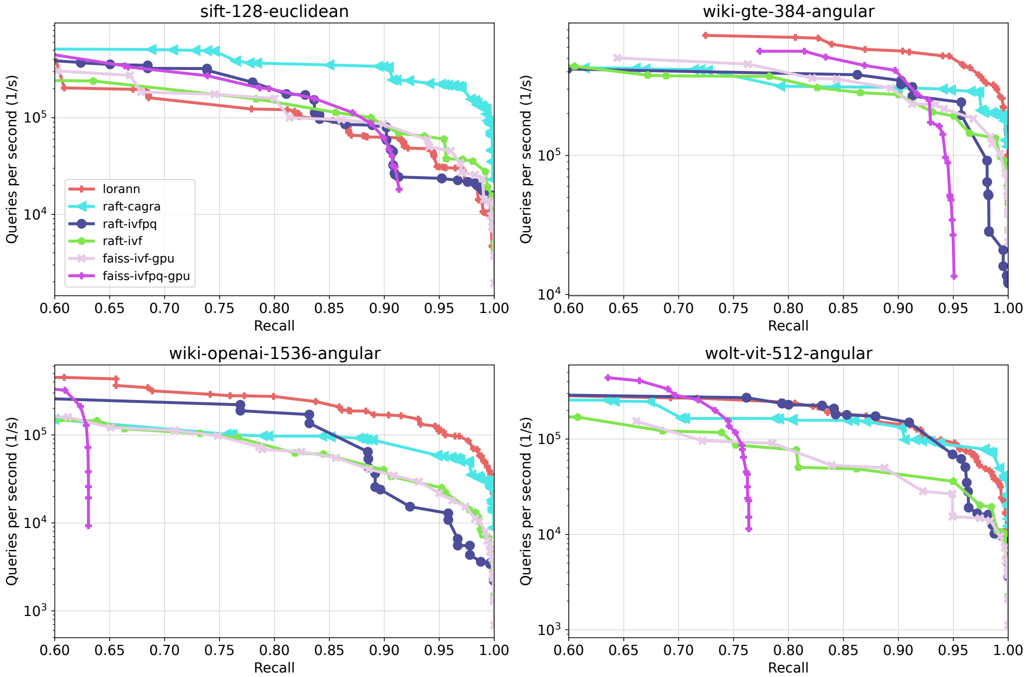
Table 2: Index construction times (seconds) at 90% recall for optimal hyperparameters.

data set	IVFPQfs	LoRANN	ScaNN	NNDesc.	QSGNGT	Glass	HNSW
ag-distilbert	29	26	66	56	1687	73	104
ag-minilm	15	18	34	47	1763	46	203
ann-arxiv	269	1442	1160	3988	16521	4241	7421
ann-t2i	34	916	125	238	3366	304	842
arxiv-instructxl	79	1189	1259	1041	18781	2956	6779
arxiv-openai	77	112	425	208	16369	405	926
fashion-mnist	15	6	30	32	1254	22	42
fasttext-wiki	60	1012	–	1241	3025	752	1414
gist	979	305	686	1203	8610	2258	6631
glove	49	–	155	1958	14259	1728	4412
mnist	15	6	31	32	1249	22	43
sift	24	91	32	361	2785	405	851
wiki-gte	24	34	39	59	2060	61	138
wiki-openai	362	67	296	137	5085	239	506
wolt-vit	286	464	563	1103	8327	874	1982
yandex-5M	589	2444	612	5218	–	2884	6599

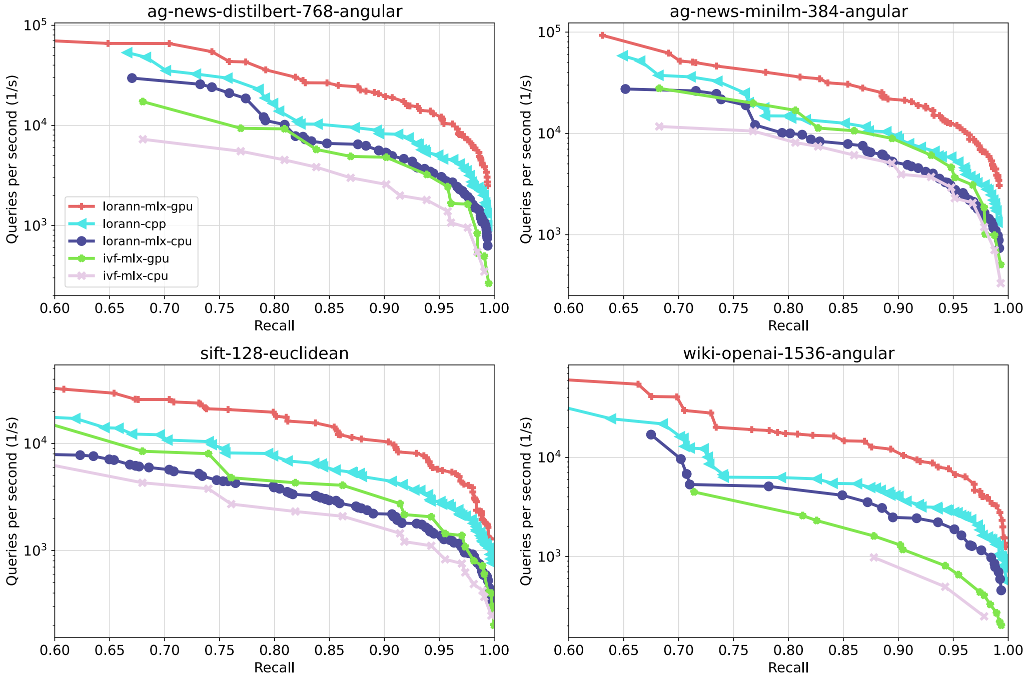
E.5 GPU evaluation

NVIDIA GPU In this section, we present the full evaluation of LoRANN in the GPU setting in comparison to other GPU ANN methods. For discussion, refer to Section 7.2.3.





Apple silicon In this section, we use MLX[8] to implement `LoRANN` for Apple silicon. We compare MLX versions (both CPU and GPU versions) of `LoRANN` against MLX versions of IVF and the C++ version of `LoRANN` (without quantization) on the Apple M2 Pro SoC. The MLX version of `LoRANN` can take advantage of the M2 Pro GPU and its unified memory architecture to achieve 2–5 times faster query latencies compared to the C++ implementation of `LoRANN`.



[8] `https://github.com/ml-explore/mlx`

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We explicitly list the main contributions of the article at the end of Section 1 and refer to the relevant sections of the article justifying the claims being made.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of the work in Section 8.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The proof of Theorem 1 is provided.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide code implementing our method as well as code for running the benchmarks, including code to prepare all the data sets and algorithms.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide code implementing our method as well as code for running the benchmarks, including code to prepare all the data sets and algorithms. The provided repository has instructions for running the experiments.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental set-up is described in Appendix B and the details for all data sets are given in Appendix C. The experimental code including data set preparation and used hyperparameters for all methods is available at `https://github.com/ejaasaari/lorann-experiments`.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: It is not customary to report error bars or test the statistical significance in the field of ANN search because it would be computationally expensive. We perform the experiments using the ANN-benchmarks framework (Aumüller et al., 2020) which is a de facto standard in the field.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We mention the exact AWS instances used in the experiments in Appendix B.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We checked that our work presented in the article conforms with the NeurIPS code of Ethics. In particular, we checked that the data sets used were not deprecated and had a permissive license (see our answer to Question 12), and we disclose the elements of reproducibility (see our answer to Question 4).

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: The article is foundational research as ANN search is a primitive that is used as a component in machine learning pipelines. Thus, our contribution of making ANN search more efficient has no direct societal impact.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The article is foundational research, so there are no elements that have a high risk for misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use data sets with open licenses and our benchmarking code downloads the used data sets from their original sources; we also list all used data sets in Appendix C along with their licenses and links. Our benchmarking code is a fork of the ANN-benchmarks project which is licensed under the MIT license.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide documentation for both `LoRANN` and our experiments in their respective repositories.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The article does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The article does not involve crowdsourcing or research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.