arXiv:2304.02350v2 [cs.IR] 6 Apr 2023

展开式自重构局部敏感哈希：面向近似近邻搜索中的机器遗忘

Kim Yong Tan[1], Yueming Lyu[2⋆]、Yew Soon Ong[1], 和 Ivor W. Tsang[2]

[1] 新加坡南洋理工大学{kimyong001,asysong}@ntu.edu.sg[2] 前沿人工智能研究中心（CFAR） 科学、技术与研究局（A*STAR）{lyu yueming,ivor tsang}@cfar.a-star.edu.sg

摘要。近似最近邻（ANN）搜索是搜索引擎、推荐系统等不可或缺的组成部分。近年来，许多工作专注于基于学习的数据分布相关哈希，并取得了良好的检索性能。然而，由于用户对用户隐私和安全性的需求日益增长，我们通常需要从机器学习（ML）模型中移除用户的数据信息，以满足特定的隐私和安全性要求 [4,20,9,6]。这一需求要求ANN 搜索算法支持快速的在线数据删除和插入。当前基于学习的哈希方法需要重新训练哈希函数，而大规模数据的巨大时间成本使这一过程难以接受。为解决这一问题，我们提出了一种新颖的数据相关哈希方法，称为展开式自重构局部敏感哈希（USR-LSH）。我们的 USR-LSH 对逐实例数据重构的优化更新进行了展开，相较于数据无关LSH，更有利于保留数据信息。此外，我们的USR-LSH无需重新训练即可支持快速的在线数据删除和插入。据我们所知，我们是首个解决检索问题中机器遗忘问题的研究者。在实证实验中，我们证明了在ANN 任务中，就精确率和召回率而言，USR-LSH 的表现优于最先进的数据分布无关 LSH。我们还表明，与基于学习的数据相关哈希相比，USR-LSH 的数据删除和插入速度显著更快。

关键词：近似近邻搜索 · 隐私 · 数据分布无关的局部敏感哈希 · 信息检索。

1 引言

给定一个数据点集合 $\boldsymbol{X} = \{x_1, x_2, ..., x_N\}$ 和一个查询点 $\boldsymbol{q}$，精确 k 最近邻（k近邻）搜索问题是找到 k 个不同的点，这些点是

⋆ 通讯作者

最接近查询点 $\boldsymbol{q}$。存在用于低维情形下精确最近邻搜索的高效算法 [2] 。在大规模高维情形下，问题变得困难，大多数算法甚至比线性扫描 [21]需要更高的计算成本。这种现象通常被称为“维数灾难” [11]。因此，近年来大量工作致力于搜索近似近邻（ANN） [11]。目标是在准确性上进行适度权衡，以显著提高效率，找到 k 与给定查询点$\boldsymbol{q}$ “足够接近”的不同点。

解决 ANN 搜索问题的一种常用算法是局部敏感哈希（LSH） [11,8]。其核心思想是以较高的概率将相似的数据点映射到相同的哈希码。二进制局部敏感哈希是局部敏感哈希的一种，可生成二进制哈希码。它可以通过对应哈希码之间的汉明距离，近似计算数据点之间的欧几里得距离或角距离。与计算欧几里得距离、余弦距离等其他距离相比，计算汉明距离更快，因为它主要涉及位运算。8 位二进制哈希码可以打包成一个字节；由于其位表示紧凑，因此能够降低存储需求。然而，数据无关 LSH 的性能有限，因为它完全依赖概率，即哈希函数与数据分布无关，且哈希码之间不存在相关性。因此，人们提出了多种基于学习的 LSH 方法 [22,19]。这些方法通过训练来学习数据集分布，从而提升性能。

如今，由于用户隐私和安全性的需求日益增长，我们经常需要从机器学习（ML）模型中删除用户的数据信息，以满足特定的隐私和安全性要求。在过去十年中，面对机器学习在个人数据上的应用规模空前扩大，许多国家已在数据隐私立法中确立了被遗忘权相关条款，要求企业在需要时删除特定个人数据 [4], 例如，欧盟的《通用数据保护条例》（GDPR） [20], 美国的《加利福尼亚州消费者隐私法》（CCPA） [9], 以及加拿大的PIPEDA隐私立法 [6]。

这就需要一种 ANN搜索算法来支持快速的在线数据删除和插入。然而，当前基于学习的哈希方法需要重新训练哈希函数，而大规模数据所需的巨大时间成本使这一过程难以承受。在本文中，我们提出了一种新颖的数据分布无关哈希方法，称为展开式自重构局部敏感哈希（USR-LSH），以解决检索问题中的机器遗忘。据我们所知，我们是首个解决这一问题的研究者。通过实验，我们证明了在ANN 任务中，就精确率和召回率而言，USR-LSH 优于当前最先进的数据无关 LSH。我们还表明，与基于学习的数据相关哈希相比，USR-LSH 的数据删除和插入速度显著更快。最后，我们建议将多探测和多哈希表用于未来的研究。

2 相关工作

2.1 SimHash

Charikar [7] 在局部敏感哈希族中提出了simHash，用于生成二进制哈希码。给定输入向量 $\boldsymbol{x} \in \mathbb{R}^d$，哈希函数如下：

$$\boldsymbol{y} = sign(\boldsymbol{W}\boldsymbol{x}) \tag{1}$$

$\boldsymbol{W} \in \mathbb{R}^{md}$ 是一个随机投影矩阵，其中 $\boldsymbol{W}$ 的每个条目都取自高斯分布，$W_{ij} \sim \mathcal{N}(0, \boldsymbol{I}_d)$。$\boldsymbol{W}$中的一行 $\boldsymbol{w_i} \in \mathbb{R}^d$ 在几何上可以看作一个随机单位向量，该向量定义了一个与 $\boldsymbol{w_i}$ 垂直并经过原点的超平面。我们在 $\boldsymbol{W}$中有 m 个这样的随机超平面。$sign$（$\boldsymbol{W}\boldsymbol{x}$）使用 m 个随机超平面对输入空间进行切分，并根据 $\boldsymbol{x}$ 位于超平面 $\boldsymbol{w_i}$的哪一侧来取符号。因此，这个哈希函数也被称为超平面局部敏感哈希。由于所有超平面都经过原点，它只能保留余弦距离，而不能保留欧几里得距离。

2.2 超级比特局部敏感哈希

Ji 等人 [12] 提出了超比特局部敏感哈希（SB-LSH），以改进随机投影哈希（simHash）。他们的后续工作 [13] 进一步从理论上证明，与 simHash 相比，对于 $(0, \pi)$ 中的任意角度，超比特局部敏感哈希都能以更小的方差对成对角相似度进行无偏估计。给定输入向量 $\boldsymbol{x}$，其中 $\mathbb{R}^d$，哈希函数如下：

$$\boldsymbol{y} = sign(\boldsymbol{W}\boldsymbol{x}), \tag{2}$$

其中 W 是一个堆叠随机旋转矩阵，其初始化如下：

$$\boldsymbol{W} = \sqrt{\frac{d}{h}}[\boldsymbol{R}_1, ..., \boldsymbol{R}_m], \tag{3}$$

其中，$\boldsymbol{R}_i \in \mathbb{R}^{d \times d}$ 是一个随机旋转矩阵，$\boldsymbol{R}_i$ 中的每一行长度均为单位长度，且彼此正交。我们将 m 个旋转矩阵堆叠起来，生成 $\boldsymbol{W}$。要生成每个 $\boldsymbol{R}_i$，我们可以生成一个高斯随机矩阵 $\boldsymbol{S} \in \mathbb{R}^{d \times d} \sim \mathcal{N}(0, \boldsymbol{I}_d)$，其元素从标准高斯分布中采样，然后对其进行正交化以生成一个 $\boldsymbol{R}_i$。正交化过程采用 Gram-Schmidt 过程，即将当前向量正交投影到由前序向量张成的子空间的正交补上。

2.3 Faiss 库

Faiss [14] 是一个最先进的近似最近邻（ANN）搜索库，采用高度优化的 C++ 代码编写，并带有 Python 封装。在其局部敏感哈希模块 faiss.IndexLSH 中，它将输入 $\boldsymbol{x} \in \mathbb{R}^d$ 哈希为哈希码 $\boldsymbol{b} \in \{-1, +1\}^n$。当 $n < d$ 时，Faiss-局部敏感哈希使用 simHash；当 $n >= m$ 时，它使用 SB-LSH。给定一个查询，faiss.IndexLSH 通过对二进制哈希码空间进行穷举搜索来查找 k近邻。

2.4 哈希码排序

给定数据集 $h(\boldsymbol{X})$ 的哈希码和查询 $h(\boldsymbol{q})$ 的哈希码，获取其最近邻的最基本方法是计算 $h(\boldsymbol{q})$ 与 $h(\boldsymbol{X})$ 之间的汉明距离，进行穷举搜索，然后选择距离最小的前 k 个点作为 k 近邻。

当数据规模较小时，穷举搜索通常比哈希表查找更快。这种策略利用了二进制哈希码的优势：计算汉明距离比计算其他距离（如欧几里得距离和余弦距离）高效得多，因为它主要涉及按位异或运算。在本研究中，我们将使用哈希码排序来评估我们的局部敏感哈希算法。

2.5 哈希表查找

使用哈希码排序时，查询速度会随数据集的大小线性增加。借助哈希表查找[22]，我们可以实现常数时间复杂度。哈希表是一种倒排索引数据结构，其中键是哈希码 $h(\boldsymbol{X})$，值是数据点 $\boldsymbol{X}$ 的引用列表。给定查询 $h(\boldsymbol{q})$ 的哈希码，如果桶存在，哈希表查找会查找 $h(\boldsymbol{q})$桶，桶中的数据 $h(\boldsymbol{q})$ 会被检索为 q 的最近邻的候选项。如果原始数据可用，通常会接着进行排序步骤，根据候选项与查询点之间的真实距离对候选项进行排序。假设我们正在查询k个最近邻（k近邻）。然后，我们返回真实距离最小的前k个候选项。如果原始数据不可用，DenseFly [17] 会构建用于哈希表查找的低维哈希码，并构建用于候选项排序的低维哈希码。

2.6 多探测

为保证哈希表查找具有高精确率，桶大小（即哈希码的维度）需要足够大。但这意味着桶冲突的概率会过低，即 $h(\boldsymbol{q})$通常会是一个空桶。为克服这一问题，[15] 采用了这样一种理念：最近邻更有可能被哈希到相近的桶中，并智能地探测（查找）靠近 $h(\boldsymbol{q})$的桶。

在这项工作中，我们专注于构建哈希函数。除了哈希函数之外，哈希表查找和多探测也有望提升查询速度。这些技术与我们的哈希函数技术正交，可以作为插件应用于我们的方法中，以进一步提升查询速度。我们将这部分工作留作未来的研究方向。

3 方法

3.1 哈希函数：展开式自重构哈希

我们开发了一种称为展开式自重构哈希局部敏感哈希（USR-LSH）的二进制哈希函数。给定一个数据点 $\boldsymbol{x} \in \mathbb{R}^d$，我们的算法将其哈希到二进制空间 $\boldsymbol{b} \in \{-1, +1\}^{md}$中，其中值 md 是哈希码的维度。由于哈希码是二进制的，它也等于哈希位数。在本文中，我们交替使用这两个术语。d 是数据点的维度。我们可以设置超参数 m 来控制哈希码维度。更高的 m 将带来更好的哈希性能（更好的数据重构效果），但会占用更多空间。

自重构目标：我们的目标是构造能够更好地保留数据信息的哈希码。为此，我们最小化如公式(4)所示的重构误差：

$$\mathcal{J}(\boldsymbol{b}) := ||\boldsymbol{W}\boldsymbol{b} - \boldsymbol{x}||_2^2 \tag{4}$$

其中，$\boldsymbol{W} \in \mathbb{R}^{d \times md}$ 是由公式（5）构造的随机投影矩阵

$$\boldsymbol{W} = \sqrt{\frac{d}{md}}[\boldsymbol{R}_1, \cdots, \boldsymbol{R}_m], \tag{5}$$

其中 $\boldsymbol{R_i}, i \in \{1, \cdots, m\}$ 表示随机正交矩阵。 $\boldsymbol{W}$可视为解码器矩阵，将哈希码 $\boldsymbol{b}$ 解码回原始数据空间。

自重构目标 $\mathcal{J}(\boldsymbol{b})$是一个难以求解的离散优化问题。为缓解这一问题，我们用 $tanh(\cdot)$函数放宽二进制约束（$sign(\cdot)$）。然后，我们优化式(6)中的近似问题

$$\widehat{\mathcal{J}}(\boldsymbol{y}) := ||\boldsymbol{W}\boldsymbol{y} - \boldsymbol{x}||_2^2 + \phi_\lambda(\boldsymbol{y}), \tag{6}$$

并带有隐式正则化 $\phi_\lambda(\boldsymbol{y})$，使得双曲正切$(\cdot)$为近端函数，即 $\phi_\lambda(\cdot)$具有高效的近端算子 $g(\cdot) = tanh(\cdot)$，使得$g(z) = \arg\min_x \frac{1}{2}(x - z)^2 + \phi_\lambda(x)$。

展开式优化：我们现在展示如何优化近似问题 $\widehat{\mathcal{J}}(\boldsymbol{y})$。将 $y_0 \in \mathbb{R}^{md}$ 记为初始哈希值，并首先将其各项初始化为零：

$$\boldsymbol{y_0} = \boldsymbol{0}. \tag{7}$$

给定输入 $\boldsymbol{x}$ 及其在时间步 t的哈希值 $\boldsymbol{y_t} \in \mathbb{R}^{md}$，更新函数 $h : (\mathbb{R}^d, \mathbb{R}^{md}) \mapsto \mathbb{R}^{md}$ 将生成下一时间步 $\boldsymbol{y_{t+1}} \in \mathbb{R}^{md}$ 的哈希值：

$$\boldsymbol{y_{t+1}} = h(\boldsymbol{x}, \boldsymbol{y_t}), \tag{8}$$

其中，更新函数 h 定义如下：

$$h(\boldsymbol{x}, \boldsymbol{y_t}) = tanh\left(\alpha \boldsymbol{W}^\top \boldsymbol{x} + (\boldsymbol{I} - \boldsymbol{W}^\top \boldsymbol{W})\boldsymbol{y_t}\right), \tag{9}$$

其中，$\boldsymbol{W}$ 是堆叠随机旋转矩阵，α 是标量系数。

公式 (9) 中的 $\boldsymbol{W}$ 初始化如下：

$$\boldsymbol{W} = \sqrt{\frac{d}{md}}[\boldsymbol{R}_1, ..., \boldsymbol{R}_m], \tag{10}$$

其中，$\boldsymbol{R}_i \in \mathbb{R}^{d\times d}$ 是一个随机旋转矩阵。$\boldsymbol{R}_i$ 的各行是正交归一的，即它们的长度均为 1，且彼此正交，即$\boldsymbol{R}_i^\top \boldsymbol{R}_i = \boldsymbol{I}_d$。我们将旋转矩阵堆叠 m 次，以生成 $\boldsymbol{W}$。为了生成每个 $\boldsymbol{R}_i$，我们可以生成一个高斯随机矩阵 $\boldsymbol{S} \in \mathbb{R}^{d\times d} \sim \mathcal{N}(0, \boldsymbol{I}_d)$，其中各元素均从标准高斯分布中采样，然后对其进行正交化，以生成一个 $\boldsymbol{R}_i$。在我们的工作中，向量正交化过程通过奇异值分解（SVD）完成。

对于公式（9）中的 α，我们通过实验证实最佳取值为：

$$\alpha = \frac{\sqrt{md}}{2\mathbb{E}_{\boldsymbol{x}\in\boldsymbol{X}}[|\boldsymbol{x}|_2]}, \tag{11}$$

其中，$\mathbb{E}_{\boldsymbol{x}\in\boldsymbol{X}}[|\boldsymbol{x}|_2]$ 通过计算整个数据集 $\boldsymbol{X} = \{\boldsymbol{x_0}, \boldsymbol{x_1}, ..., \boldsymbol{x_N}\}$的平均欧几里得范数得到。此举旨在将每个输入点的长度缩放至接近 1，从而改进实例级学习。

我们迭代更新函数 h，经过 T 时间生成最终哈希值$\boldsymbol{y_T}$。最后，通过哈希值的符号获得二进制哈希码 $\boldsymbol{b} \in \{-1, +1\}^{md}$：

$$\boldsymbol{b} = sign(\boldsymbol{y_T}), \tag{12}$$

其中，符号函数定义为：

$$sign(x) = \begin{cases} -1 & \text{if } x < 0 \\ +1 & \text{if } x \geq 0 \end{cases}. \tag{13}$$

我们将 T步更新展开为USR-LSH 网络架构。用于优化 $\mathcal{J}(\boldsymbol{y})$ 的算-$\widehat{}$法见算法 1。给定一个近端函- 数，[16] 从理论上证明了重构误差会降低，即，$\widehat{} \sim \mathcal{J}(\boldsymbol{y}_t) \leq \mathcal{J}(\boldsymbol{y}_{t-1})$，对于每次迭代 $t = \{1, \cdots, T\}$均如此。本文采用 $tanh(\cdot)$函数作为近端函数，通过隐式正则化来构- 建哈希码。此外，在实践中，可以对一批输入点 $\boldsymbol{X}$ 并行执行该操作。

算法1 USR-LSH

输入：数据 x、哈希码长度 m、迭代次数 T、标量系数 α。
初始化： $\boldsymbol{y}_0 \leftarrow \mathbf{0}$
对于 t= 1 至 T，执行
  设置 $\boldsymbol{y}_t \leftarrow tanh\left(\alpha \boldsymbol{W}^\top \boldsymbol{x} + (\boldsymbol{I} - \boldsymbol{W}^\top \boldsymbol{W})\boldsymbol{y}_{t-1}\right)$
结束循环
返回 $sign\left(\boldsymbol{y}_T\right)$

3.2 哈希桶探测

使用哈希码排序时，计算成本会随着数据规模线性增长。另一方面，使用哈希表查找时，哈希码足够长后，发生哈希碰撞的概率会变得过低。为此， [18] 采用了这样的思路：最近邻更有可能被哈希到彼此相邻的桶中，并智能地探查（查找）靠近 h（$\boldsymbol{q}$）的桶。

给定哈希码 $g(\boldsymbol{q}) = (h_1(\boldsymbol{q}), ..., h_M(\boldsymbol{q}))$，其中 $g(\boldsymbol{q}) \in [0, b]^M$，多探针局部敏感哈希 [15] 将哈希扰动向量定义为 $\boldsymbol{\Delta} = (\delta_1, ..., \delta_M)$，其中 $\delta_i \in \{-1, 0, +1\}$。然后，我们应用扰动向量来探测附近的桶 $h(\boldsymbol{q}) + \boldsymbol{\Delta}$。我们为每个扰动 $\boldsymbol{\Delta_i}$分配一个分数，分数越小，产生靠近 $\boldsymbol{q}$的点的概率越高。注意， $\boldsymbol{\Delta}$ 的分数是 $\boldsymbol{\Delta}$ 和查询 $\boldsymbol{q}$的函数。然后，我们按照（依赖于查询的）分数递增的顺序排列扰动向量，生成扰动向量序列 $(\boldsymbol{\Delta_1}, \boldsymbol{\Delta_2}, ..., \boldsymbol{\Delta_n})$，以访问位于 $(h(\boldsymbol{q}) + \boldsymbol{\Delta_1}, h(\boldsymbol{q}) + \boldsymbol{\Delta_2}, ..., h(\boldsymbol{q}) + \boldsymbol{\Delta_n})$ 的 n 个附近桶。这就是 [15]中描述的查询导向探测方法。

在 USR-LSH 中，我们的哈希码是二进制的， $h(\boldsymbol{q}) \in \{+1, -1\}^{md}$，因此我们不进行加法（或减法），而是使用扰动向量来翻转哈希码中某些坐标的符号。我们可以将扰动向量定义为 $\boldsymbol{\Delta} = (\delta_1, ..., \delta_M)$，其中 $\delta_i \in \{-1, +1\}$。我们不将其加到哈希码上，而是进行逐元素乘法来翻转哈希码的符号， $h(\boldsymbol{q}) \odot \boldsymbol{\Delta}$。也就是说，我们将按以下序列探测相邻的桶：$(h(\boldsymbol{q}) \odot \boldsymbol{\Delta_1}, h(\boldsymbol{q}) \odot \boldsymbol{\Delta_2}, ..., h(\boldsymbol{q}) \odot \boldsymbol{\Delta_n})$。

我们的哈希值 $\boldsymbol{b} \in \mathbb{R}^{md}$ 在L2距离下重构数据点，因此，最接近的扰动是翻转最接近0的坐标。基于这一思想，我们可以将分数定义为编码 b的扰动坐标绝对值之和：

$$score(\boldsymbol{\Delta}) = \sum_{i=0}^{md} (\frac{1 - \delta_i}{2} |b_i|), \tag{14}$$

其中， $\boldsymbol{b_i}$ 是哈希值 $\boldsymbol{b}$的第 i个坐标。请注意，扰动向量 $\boldsymbol{\Delta}$ 的得分仅取决于其经过扰动的坐标 $\delta_i = -1$。我们

预计低分的扰动向量在大多数条目中将具有 $+1$ 。

使用此评分函数，我们可以像在 [15] 中一样，利用一个最小堆生成探测序列。我们还可以轻松地将其推广到多个哈希表，以进一步提高精确率。

4 实验

4.1 数据集

我们在来自知名近似最近邻基准测试框架 [1]的四个数据集上评估每种算法，如表 1 所示。我们遵循 [1] 对角度数据集进行欧几里得归一化。

表 1：实验中使用的数据集。

数据集	Size	维度	距离
Deep1B	9,990,000	96	角距离
SIFT	1,000,000	128	欧几里得距离
GloVe-100	1,183,514	100	角距离
Glove-25	1,183,514	25	角距离

4.2 符号与定义

给定一个查询点 $\boldsymbol{q}$， Y 表示 $\boldsymbol{q}$ 的真实最近邻ˆ ˆ（NN）集合。假设我们正在查询 k 个最近邻点（k-查询），ˆ并将查询结果集合表示为 Y，我们可以假设该算法始终返回 ˆk 个点，即ˆ $k = |\hat{Y}|$。

ˆ ˆ k-查询结果的精确率@k 定义为查询结果中正确点所占的百分比。其定义如下：

$$precision@\hat{k} = \frac{|\hat{Y} \cap Y|}{|\hat{Y}|}. \tag{15}$$

ˆ ˆ k查询结果的召回率@k 定义为查询结果中所返回的真实最近邻点 Y 所占的百分比。其定义如下：

$$recall@k = \frac{|\hat{Y} \cap Y|}{|Y|}. \tag{16}$$

在我们的数据集中，对于任意查询，我们都有 10 个真实值点，即 $|Y| = 10$。ˆ k 在不同的实验中有所不同。在实验 1 中，我们查询 10 ˆ 个最近邻，即 $k = k = 10$。在这种情况下，Precision@10 等于召回率@10。随后，我们评估查询时间与 Precision@10 之间的权衡。ˆ 在实验 2 中，我们查询 $k = \{1, ..., 100\}$ 个最近邻，并评估 ˆ 每个 k 的精确率与召回率之间的权衡。实验 3 用于评估 ˆ 删除时间，该时间不受 k 或 k 的影响。

4.3 实验设置

我们的实验基于 Martin 的 ANN-benchmark 框架 [1]。所有实验均使用完整数据集作为输入，并在 1000 个查询点上进行评估，以计算平均召回率和精确率。所有实验均在 Intel(R) Xeon(R) Gold 6342 中央处理器 @ 2.80GHz 上运行（该中央处理器每个核心有一个线程）。通过使用命令 taskset，将所有实验运行限制在单核中央处理器上，并使用一个线程。复现我们实验的源代码可在匿名 GitHub [3] 上找到。

4.4 实验1：每秒查询数 - Precision@10 权衡

在本实验中，我们评估了每种算法的每秒查询数（QPS）与 Precision@10 之间的权衡。给定一个查询 $\boldsymbol{q}$，我们获取其哈希码ˆ$h(\boldsymbol{q})$，并对数据集哈希码 $h(X)$ 执行穷举搜索，以查找其 k 个最近邻。我们每次只执行一个查询，不进行并行化。

真实值数据集为每个查询点 $\boldsymbol{q}$包含10个最近邻点，且 $k = 10$；我们对每个查询ˆ ˆ查询了10个最近邻，且 $k = 10$。请注意，在本例中，我们设置 $k = k$，因此召回率与精确率的值是等价的，即精确率@10 = 召回率@10。

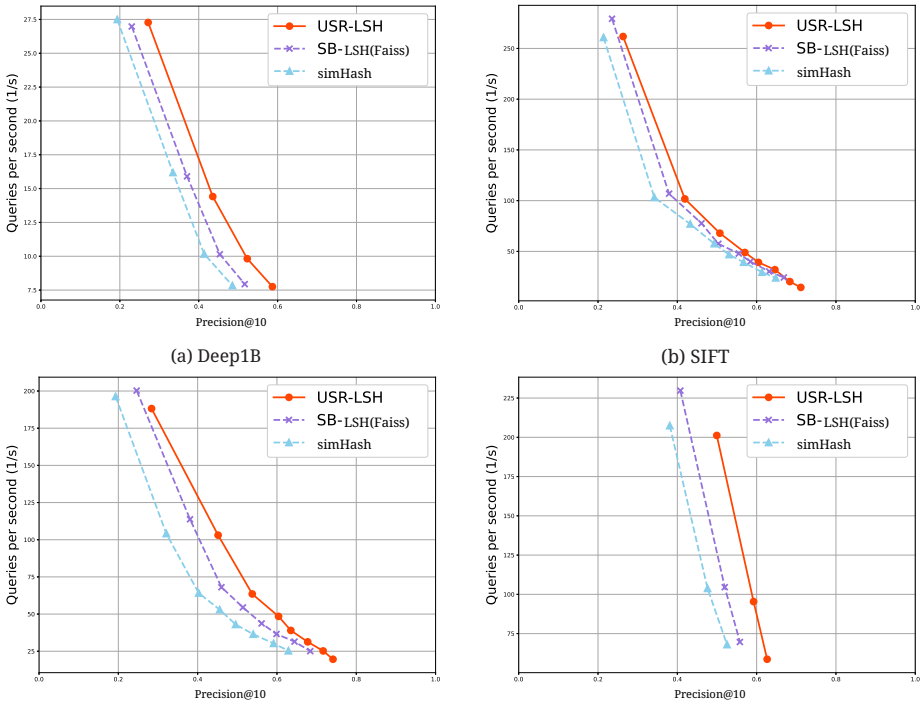
我们针对每种算法在不同参数设置下进行了实验，并绘制了每种算法的帕累托前沿。我们设置参数，使每种算法具有相同的哈希位数。例如，如果我们为simHash和USR-LSH设置m ，则会将SB-LSH参数设置为n位 $= md$，其中 d 为数据维度，从而使哈希位数相同。表 2展示了本实验中使用的参数。

图 1 展示了各数据集的每秒查询数和 Precision@10 曲线。我们的 USR-LSH 在所有数据集上的表现都优于 simHash 和 SB-LSH。可以看出，USR-LSH 的查询时间略慢于其他方法，因为它会迭代 T 次，但精确率的提升使其帕累托前沿超越了 simHash。

[3] https://anonymous.4open.science/r/ann-benchmarks-3786

和 SB-LSH。

请注意，SB-LSH 是在官方 Faiss 库中实现的，该库采用高度优化的 C++ 代码编写。对于 simHash 和（我们提出的）USR-LSH，哈希函数使用 Python 实现，并通过 JAX 的即时编译器 [5] 预编译。随后，哈希值会传递给 Faiss 库，以进行位打包和哈希码汉明排序，从而查找 k近邻。如果进一步使用 C++ 代码优化 USR-LSH，我们可以期待更快的查询速度。



(a) Deep1B (b) SIFT

(c) GloVe-100 (d) Glove-25

图 1：每秒查询数（1/s）与 Precision@10 之间的权衡

4.5 实验 2：精确率-召回率权衡

在本实验中，我们评估每种算法的精确率-召回率权衡。真实最近邻的数量为 $|Y| = k = 10$。给定查询 ˆ 点 $\boldsymbol{q}$，我们搜索其 1 到 100 个最近邻，即对 ˆ ˆ$k = 1$, ..., 100 执行 k-查询。然后，我们评估每个 k-查询的精确率和召回率指标，以生成精确率-召回率（PR）曲线。最后，我们计算精确率-召回率曲线下的面积（PR曲线下面积）以评估性能，PR曲线下面积越高

表 2：实验 1 中每个数据集均使用全部算法参数。实验 2 使用最低和最高（带下划线）参数。

数据集	参数		
	simHash（m=）	SB-LSH（n_位=）	USR-LSH (T=17,m=)
Deep1B	{2,3,4,5,6,7,8}	{192,288,384,480,576,672,768}	{2,3,4,5,6,7,8}
SIFT	{2,4,6,8,10,12,16,20}	{256,512,768,1024,1280,1536,2048,2560}	{2,4,6,8,10,12,16,20}
GloVe-100	{2,4,6,8,10,12,16,20}	{200,400,600,800,1000,1200,1600,2000}	{2,4,6,8,10,12,16,20}
GloVe-25	{2,4,6,8,10,12,16,20}	{50,100,150,200,250,300,400,500}	{2,4,6,8,10,12,16,20}

意味着性能越好。

我们在所有数据集（表 1）上运行了实验。我们在低哈希码维度和高哈希码维度设置下评估算法性能（表 2 中带下划线的参数）。与实验 1 相同，这些参数的设置使得各算法在相同数量的哈希码下进行比较。

图 2 和图 3 分别展示了各算法在所有数据集上使用低维和高维哈希码时的PR曲线。可以观察到，在每个数据集中，无论采用低维还是高维哈希码设置，我们的 USR-LSH 都优于其他算法。

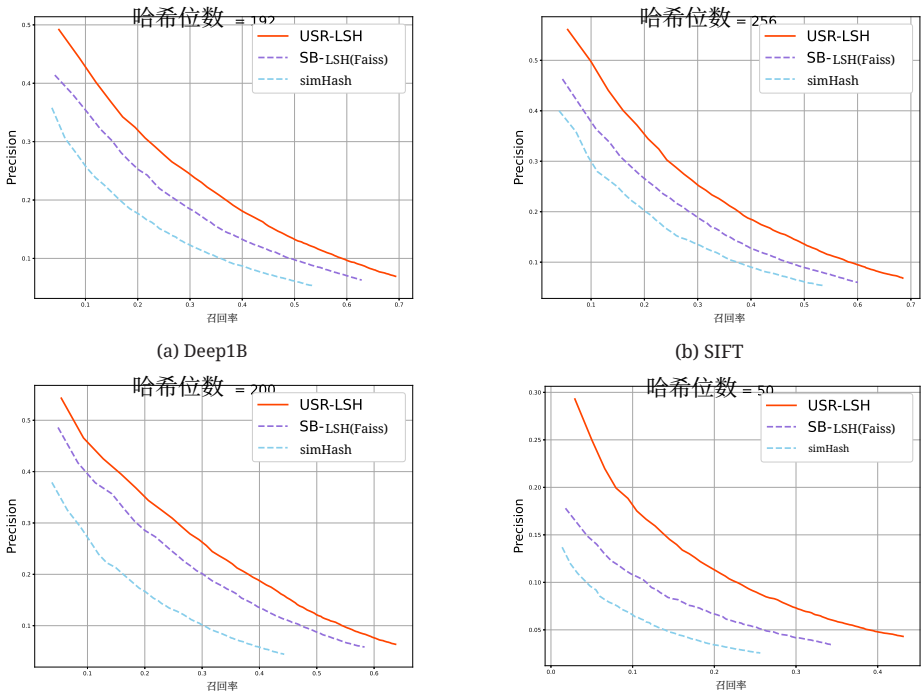
表 3展示了精确率-召回率曲线下面积。我们使用simHash作为基线，并比较SB-LSH和我们的 USR- LSH在每个数据集上的曲线下面积提升幅度（百分比）。平均而言，在低维哈希码设置下，我们的 USR-LSH 提升超过131%，在高维哈希码设置下提升 28%。与SB-LSH相比，在低维哈希码下，USR-LSH实现了超过两倍的提升（+62% → +131%）；在高维哈希码下，提升幅度接近三倍（+10% → +28%）。

表3：与simHash基线的PR曲线下面积比较。

算法	哈希维度	数据集				平均改进
		Deep1B	SIFT	GloVe-100	Glove-25	
USR-LSH	low	0.14(+92%)	0.15(+82%)	0.14(+123%)	0.05(+225%)	**+131%**
	high	0.55(+41%)	0.68(+13%)	0.74(+25%)	0.61(+33%)	**+28%**
SB-LSH	low	0.11(+49%)	0.11(+34%)	0.11(+78%)	0.03(+86%)	+62%
	high	0.45(+16%)	0.63(+4%)	0.66(+12%)	0.50(+10%)	+10%
simHash	low	0.08	0.08	0.06	0.01	(基线)
	high	0.39	0.61	0.59	0.45	(基线)

4.6 实验 3：数据删除时间

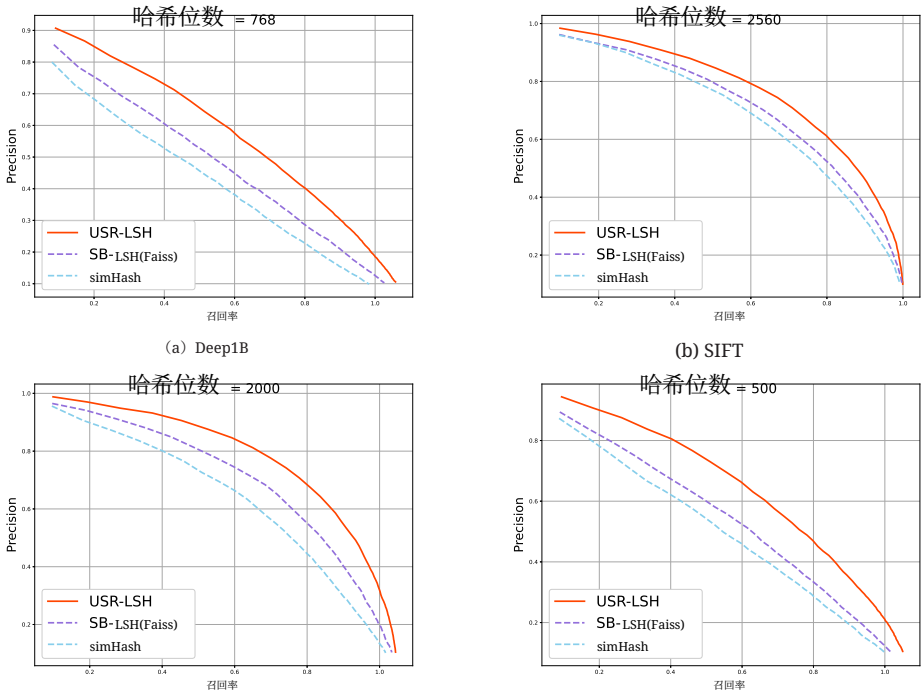
在本实验中，我们将数据删除时间与一些流行的数据相关 ANN 搜索算法进行比较：Scann [10], Annoy [3], 以及表现最佳的



(a) Deep1B (b) SIFT

(c) GloVe-100 (d) Glove-25

图 2：低维哈希码设置下的精确率-召回率曲线。



（a）Deep1B (b) SIFT

(c) GloVe-100 (d) Glove-25

图3：采用高维哈希码设置时的精确率-召回率曲线。

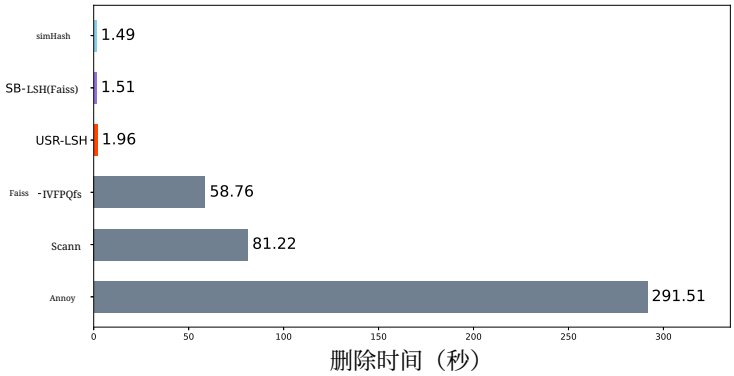


图 4：平均删除时间（秒）。

Faiss 中的模块 Faiss-IVFPQfs [14]。我们删除 10 个数据点，每次删除一个，并测量平均删除时间。对于 Scann、Annoy 和 Faiss-IVFPQfs，我们在每次删除后重新训练数据集。对于我们的 USR-LSH，我们在每次删除后重新计算数据集的平均欧几里得范数 $\mathbb{E}_{\boldsymbol{x}\in\boldsymbol{X}}[|\boldsymbol{x}|_2]$ ，以设置系数 α。

我们在最大的数据集 Deep1B 上进行实验。数据依赖方法通常性能更好，因此为了公平比较，我们选用 [1] 中最低的参数设置（可实现最快的训练时间）。对于 simHash、SB-LSH 和 USR-LSH，我们采用表 2 中 Deep1B 的最高参数设置（即哈希位数 $= 768$）。

图 4展示了每种算法的平均删除时间（以秒计）。观察到，与数据相关算法（Scann、Annoy 和Faiss-IVFPQfs）相比，数据无关算法（simHash、SB-LSH和 USR-LSH）的删除速度显著更快。我们的 USR-LSH 由于需要计算数据集的平均L2 范数，因此比 simHash和SB-LSH稍慢，但与数据相关算法的重新训练时间相比，这点差异可以忽略不计。

5 结论

本文提出了一种新颖的数据分布依赖哈希方法，称为展开式自重构局部敏感哈希（USR-LSH），以解决检索任务中的机器遗忘问题。该算法的理论依据见 [16] 。我们的实证结果表明，在精确率和召回率方面，该算法优于当前最先进的数据无关 LSH；与其他数据依赖型近似最近邻算法相比，其数据删除时间也更短。我们还建议在未来的工作中采用多探测和多哈希表。

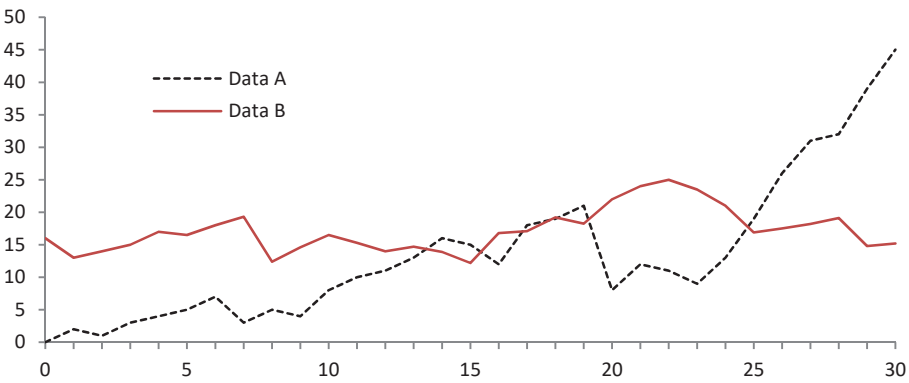
6 伦理声明

本研究使用的数据集已公开，且不包含个人信息。我们的工作提出了一种快速近邻搜索技术，不会对警务和军事领域产生直接影响。

参考文献

1. Aum¨uller,M., Bernhardsson, E.,Faithfull, A.: Ann-benchmarks：近似最近邻算法的基准测试工具。Information Systems**87**，101374（2020）2.Bentley, J.L.：用于关联搜索的多维二叉搜索树。ACM 通讯 **18**（9），509–517（1975）3.Bernhardsson, E.：Annoy：近似最近邻，没错（2017年），https://github.com/spotify/annoy 4.Bourtoule, L.、Chandrasekaran,V.、Choquette-Choo, C.A.、Jia,H.、Travers, A.、Zhang,B.、Lie, D.、Papernot, N.：机器遗忘。载于：2021年 IEEE安全与隐私研讨会（SP）。第141–159页。IEEE（2021年）5. Bradbury, J.、Frostig, R.、Hawkins, P.、Johnson, M.J.、Leary, C.、Maclau- rin, D.、Necula, G.、Paszke, A.、VanderPlas,J.、Wanderman-Milne, S.、Zhang, Q.：JAX：Python 的可组合变换+NumPy 程序（2018年），http://github.com/google/jax 6.加拿大隐私专员办公室，O.：公告：隐私专员寻求联邦法院就加拿大人网络声誉的关键问题作出裁定´网络声誉，https://www.priv.gc.ca/en/opc-news/news-and-announcements/2018181010 7/an 。Charikar,M.S.：舍入算法中的相似度估计技术。载于：第三十四届ACM 计算理论研讨会论文集。第380–388页（2002）8.Gionis, A.、Indyk,P.、Motwani, R. 等：通过哈希实现高维相似性搜索。载于：VLDB。第99卷，第518–529页（1999）9.Goldman, E.：《加利福尼亚州消费者隐私法》（CCPA）简介。SantaClara Univ.法律研究论文（2020）10. Guo,R.、Sun,P.、Lindgren,E.、Geng, Q.、Simcha,D.、Chern,F.、Kumar, S.：利用各向异性向量量化加速大规模推理。载于：国际机器学习会议（202010396 111908），https://arxiv.org/abs/ 。。Indyk,P.、Motwani, R.：近似最近邻：致力于消除维数灾难。载于：第三十届 ACM 计算理论研讨会论文集。第604–613页（1998）12.Ji, J.、Li,J.、Yan,S.、Zhang,B.、Tian, Q.：超比特局部敏感哈希。神经信息处理系统进展 **25**（2012年）13. Ji, J.、Yan, S.、Li, J.、Gao,G.、Tian, Q.、Zhang, B.：用于角相似度的批量正交局部敏感哈希。IEEE 模式分析与机器智能汇刊 **36**（10），1963–1974（2014）14. Johnson, J.、Douze, M.、J´egou,H.：利用图形处理器进行十亿规模的相似性搜索。IEEE 大数据汇刊 **7**（3），535–547（2019）15. Lv, Q.、Josephson,W.、Wang, Z.、Charikar,M.、Li,K.：多探针LSH：用于高维相似性搜索的高效索引。载于：第33届超大型数据库国际会议论文集。第950–961页（2007）

16.Lyu, Y.,Tsang, I.：神经优化核：迈向稳健的深度学习。arXiv预印本 arXiv:2106.06097（2021年）17.Sharma,J., Navlakha, S.：利用高维局部敏感哈希改进相似性搜索。arXiv预印本arXiv:1812.01844（2018年） 18. Tao, Y.,Yi, K., Sheng,C.,Kalnis, P.：高维最近邻搜索的质量与效率。载于：2009年ACMSIGMOD数据管理国际会议论文集。第563–576页（2009年） 19. Thrun, S., Pratt, L.：学会学习。SpringerScience &Business Media（2012年） 20.Voigt, P., Von dem Bussche, A.：欧盟通用数据保护条例（GDPR）。实用指南，第1版，Cham：Springer International Publishing 10(3152676)，10–5555（2017年） 21. Wang,J.,Shen,H.T., Song,J., Ji,J.：用于相似性搜索的哈希：综述。arXiv预印本 arXiv:1408.2927（2014年） 22. Wang, J., Zhang,T.,Sebe, N., Shen,H.T.等：关于学习哈希的综述。IEEE模式分析与机器智能汇刊 40(4)，769–790（2017年）


50
45
40
35
30
25
20
15
10
5
0
0
5
10
15
20
25
30
Data A
Data B

此图“glove-100-angular.png”以“PNG”格式提供，来源为：

http://arxiv.org/ps/2304.02350v2

此图“sift-128-euclidean.png”以“PNG”格式提供，来源：

http://arxiv.org/ps/2304.02350v2