软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(3):1021−1036 [doi: 10.13328/j.cnki.jos.007518] [CSTR: 32375.14.jos.007518]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

GoVector: I/O-高效的高维向量近邻查询缓存策略*

周依杰, 林圣原, 巩树凤, 余 松, 范书豪, 张岩峰, 于 戈

(东北大学 计算机科学与工程学院, 辽宁 沈阳 110819)

通信作者: 巩树凤, E-mail: gongsf@mail.neu.edu.cn

摘 要: 基于图结构的高维向量索引 (索引图) 因其高效的近似最近邻搜索能力, 已成为大规模向量检索的主流方法. 索引图执行近似最近邻搜索 (approximate nearest neighbor search, ANNS) 的过程分为两个阶段: 第 1 阶段从入口点出发快速定位到查询向量附近区域; 第 2 阶段在查询向量附近搜索离其最近的 k 个向量. 然而, 由于索引图需存储大量邻接关系, 导致内存开销大, 因此实际部署时通常需将其存储于外存. 当执行近似最近邻搜索时, 按需加载索引图和向量数据会导致频繁发生 I/O 操作, 并成为检索性能的主要瓶颈 (I/O 时间占 90% 以上). 现有系统利用入口点及其附近邻居被高频访问的特性, 采用静态缓存策略将入口点及其若干跳邻居预先缓存在内存中, 以减少第 1 阶段的 I/O 访问. 然而分析发现, 第 2 阶段为了获取更高精度的检索结果, 需访问大量与查询向量相关的图顶点, 成为 I/O 开销的主要来源. 由于第 2 阶段的访问顶点随查询向量动态变化, 现有静态缓存策略难以有效命中, 导致其在此阶段几乎失效. 针对此问题, 设计了一个静态-动态混合缓存策略 GoVector, 其核心设计体现在: (1) 静态缓存区预加载入口点及其高频近邻; (2) 动态缓存区自适应地缓存第 2 阶段中空间局部性高的顶点. 为了进一步适配第 2 阶段中以向量相似性为导向的搜索过程, 设计了基于向量空间相似性磁盘布局策略, 通过重排顶点存储顺序, 使相似向量在物理存储上聚集于相同或相邻磁盘页, 从而显著提升数据访问的局部性. 这种双重优化机制使得缓存命中率得到显著提升, 有效降低了整体 I/O 开销. 在多个公开数据集上的实验结果表明, 当召回率为 90% 时, 相较于当前最先进的基于磁盘的索引图系统, GoVector 实现 I/O 次数平均降低 46%、查询吞吐率提升 1.73 倍、延迟下降 42%.

关键词: 高维向量; 近似最近邻搜索; 索引图

中图法分类号: TP311

中文引用格式: 周依杰, 林圣原, 巩树凤, 余松, 范书豪, 张岩峰, 于戈. GoVector: I/O-高效的高维向量近邻查询缓存策略. 软件学报, 2026, 37(3): 1021–1036. http://www.jos.org.cn/1000-9825/7518.htm

英文引用格式: Zhou YJ, Lin SY, Gong SF, Yu S, Fan SH, Zhang YF, Yu G. GoVector: I/O-efficient Caching Strategy for High-dimensional Vector Nearest Neighbor Search. Ruan Jian Xue Bao/Journal of Software, 2026, 37(3): 1021–1036 (in Chinese). http://www.jos.org.cn/1000-9825/7518.htm

GoVector: I/O-efficient Caching Strategy for High-dimensional Vector Nearest Neighbor Search

ZHOU Yi-Jie, LIN Sheng-Yuan, GONG Shu-Feng, YU Song, FAN Shu-Hao, ZHANG Yan-Feng, YU Ge

(School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: Graph-based indexes for high-dimensional vectors have become the mainstream solution for large-scale approximate nearest neighbor search (ANNS) due to their high efficiency. The search process over graph-based indexes typically consists of two stages: the

* 基金项目: 国家自然科学基金 (U2241212, 62461146205); 国家社会科学基金 (21&ZD124); 中央高校基本科研业务费专项资金 (N2116005, N2116008, N2416003); CCF-华为胡杨林基金
周依杰和林圣原为共同第一作者.
本文由“向量数据库及 DB4LLM 技术”专题特约编辑高宏教授、李国良教授、张蓉教授推荐.
收稿时间: 2025-05-06; 修改时间: 2025-06-30; 采用时间: 2025-08-20; jos 在线出版时间: 2025-09-02
CNKI 网络首发时间: 2026-01-15

first stage rapidly navigates from an entry point to a region near the query vector, while the second stage searches for the k nearest vectors within the localized region. However, due to the need to store a large number of adjacency relationships, graph-based indexes often incur high memory overhead. In practice, this leads to storing the index in external memory, where vector and graph data are loaded on demand during ANNS. This results in frequent I/O operations, which have become the primary bottleneck, accounting for over 90% of total query time. Existing systems exploit the fact that entry points and their nearby neighbors are frequently accessed, and adopt static caching strategies that preload these points and their multi-hop neighbors into memory to reduce I/O during the first stage. However, this study finds that the second stage contributes the majority of I/O cost, as it involves accessing a large number of graph vertices related to the query vector to ensure high recall. Since the accessed vertices in this stage vary dynamically with each query, static caching strategies fail to capture them effectively and thus become nearly ineffective. To address this issue, a hybrid caching strategy termed GoVector is proposed, which integrates both static and dynamic components. Specifically, (1) the static cache preloads the entry point and its frequently accessed neighbors, while (2) the dynamic cache adaptively stores high-locality vertices encountered during the second stage of the search. Furthermore, to align with the similarity-driven search behavior of the second stage, a vector-similarity-aware disk layout strategy is proposed, which reorganizes the storage order of vertices to cluster similar vectors into the same or adjacent disk pages, thus enhancing data locality. This dual-optimization approach significantly improves cache hit rates and effectively reduces overall I/O overhead. Experimental results on multiple public datasets demonstrate that, under 90% recall, GoVector achieves an average of 46% fewer I/O operations, 1.73× higher query throughput, and 42% lower latency compared to state-of-the-art disk-based graph indexing ANNS systems.

Key words: high-dimensional vector; approximate nearest neighbor search (ANNS); graph-based index

随着大语言模型 (large language model, LLM) 等生成式人工智能技术的迅速发展, 向量化语义检索在自然语言处理[1,2]、信息检索[3,4]和推荐系统[5,6]等多个应用中起到关键作用. 特别是在检索增强生成 (retrieval-augmented generation, RAG) 技术中, 向量语义检索已成为提升生成质量与响应效率的关键机制[7]. 为了应对高维向量检索的复杂性[8], 近似最近邻搜索 (approximate nearest neighbor search, ANNS) 技术得到广泛应用, 其中基于图的索引方法, 如 HNSW[9]、NSG[10]、DiskANN[11]等凭借低延迟、高精度与高吞吐量的特点, 受到学术界和工业界的广泛关注[12−15]. 然而, 索引图需显式存储大规模向量之间的邻接关系, 导致索引本身消耗大量存储空间. 随着向量规模不断扩大, 将全部索引常驻于内存变得代价高昂, 因此越来越多的系统采用基于磁盘的 ANNS 方案, 将索引图部分或全部存储在磁盘中, 以降低内存压力并提升系统可扩展性[11,16]. 其中, 以 DiskANN 为代表的基于磁盘的索引图技术被广泛集成于业界高性能向量数据库系统 (如 Pinecone[17]、Milvus[18]) 中, 以支持千亿规模向量数据的高效检索. 这类方法的核心是从入口顶点开始拓展, 每次从磁盘中加载拓展顶点所在的磁盘页 (page), 以获取其向量信息及邻接顶点, 并根据邻接顶点与查询向量的距离选择下一次的拓展顶点.

然而, 这种基于磁盘的索引图方法普遍面临严重的 I/O 性能瓶颈. 我们在 5 个真实数据集[8]上对当前最先进的两个基于磁盘的 ANNS 系统 (即 DiskANN 和 Starling[16]) 的查询性能进行了详细分析. 图 1 展示了在 90% 的召回率下, 两个系统在不同数据集上的查询过程中, I/O 操作与 CPU 计算所占总耗时的比例. 实验结果表明, I/O 操作在整体查询延迟中占据主导地位, 在 DiskANN 系统中, I/O 操作平均占比高达 83%, 在 Starling 系统中达到 79%. 这一结果说明, 磁盘访问已成为制约系统整体性能的主要瓶颈.

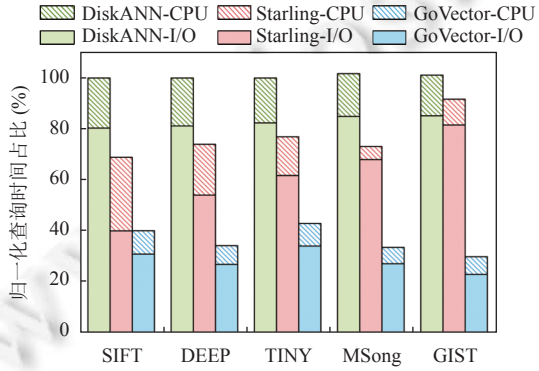


图 1 ANNS 系统查询阶段 I/O-CPU 时间占比及查询时间对比

造成该瓶颈的原因主要体现在两个方面. 一方面, ANNS 查询路径具有高度的查询相关性, 其搜索过程依赖于查询向量在向量空间中的位置, 导致实际访问的顶点在查询发起前难以准确预测. 因此, 当前主流系统普遍采用静态缓存策略, 即在查询启动前将入口顶点及其若干跳邻居提前加载到内存缓存中, 期望在搜索过程中能够直接命中这些缓存顶点, 从而减少磁盘读取次数[11]. 然而, 该策略缺乏对具体查询路径的感知能力, 导致缓存中的大量数据在实际搜索过程中未被访问, 整体命中率较低. 以 DiskANN 为例, 在进入查询向量邻近区域后, 进行 Top-k 相似向量搜索时的缓存命中率仅为 4%–9%. 另一方面, 现有系统通常通过索引图的拓扑结构来优化磁盘数据布局, 以降低 ANNS 查询过程中的 I/O 开销. 例如, Starling[16]作为当前最先进的磁盘索引图局部性优化方案之一, 通过将顶点及其拓扑邻居划分至同一磁盘页来减少随机 I/O 操作. 但该方法忽略了基于索引图查询的特殊性, 即查询过程中拓展节点的选择并非严格遵循拓扑结构的广度或深度优先顺序, 而是根据与查询点的距离来决定候选扩展顶点.

综上所述, 本文总结了当前基于磁盘的索引图方法在降低 I/O 开销方面面临的两大挑战.

(1) 缓存命中率低. 由于查询路径具有查询相关性与动态性, 现有静态缓存策略无法感知查询路径, 难以准确命中查询路径中的顶点. 需设计一种灵活且查询自适应的缓存机制, 能够根据查询向量的特性, 有选择性地缓存在搜索过程中可能被访问的关键顶点, 从而提高缓存命中率, 减少不必要的磁盘访问.

(2) 单次 I/O 的数据利用率不足. 现有以图拓扑为基础的索引布局方式, 无法充分反映实际查询路径中的访问规律, 导致每轮磁盘加载中仅有少部分数据被有效利用. 因此, 需结合 ANNS 的搜索特性, 重新设计索引的物理组织方式, 提升索引图访问的局部性, 使单次 I/O 能加载更多与当前查询相关的向量和邻接信息, 从而提升整体检索性能.

针对上述两项挑战, 本文提出 GoVector, 一种 I/O 高效的基于磁盘索引图的高维向量近邻查询缓存策略, 旨在通过查询感知的混合缓存和基于向量相似性的索引图重排策略来提升基于磁盘的索引图系统的查询效率. 本文的主要贡献如下.

(1) 提出了一种静态-动态混合缓存策略. 其中, 静态部分用于预加载入口顶点及其若干跳邻居, 以快速定位至查询向量所在的近邻区域; 动态部分用于自适应地缓存查询路径上的候选顶点及其在向量空间中相似的邻近顶点, 以提升相似向量搜索阶段的缓存命中率.

(2) 设计了一种基于向量相似性的索引布局优化策略, 通过将相似度高的向量集中存储在同一或相邻的磁盘页中, 提升了单次 I/O 中的有效数据量, 从而减少 I/O 次数, 优化整体磁盘访问性能.

(3) 在多个公开数据集上开展了系统性评估, 结果表明: 与现有方法相比, GoVector 将 I/O 次数平均减少 46% (最高 57%), 查询吞吐率提升 1.73 倍 (最高 2.25 倍), 查询延迟降低 42% (最多 55%).

本文第 1 节介绍基础知识. 第 2 节介绍研究背景与相关工作. 第 3 节介绍 GoVector 的方法概览与系统架构. 第 4、5 节介绍 GoVector 的核心设计: 混合缓存机制与索引图重排序. 第 6 节给出实验设置与性能评估结果. 第 7 节对全文进行总结.

1 基础知识

1.1 ANNS 问题定义

近似最近邻搜索[10,19,20]的目标是在一个高维向量集合中, 快速找到与给定查询向量最相似的 Top-k 向量[16]. 给定一个包含 n 个高维向量的数据集 $V=\{v_1,v_2,\ldots,v_n\}\subset\mathbb{R}^d$, 以及一个查询向量 $q\in\mathbb{R}^d$, ANNS 的目标是在 V 中找到与 q 距离最近的 k 个向量, 即返回集合:

$$\text{Top-k}(q)=\underset{R\subseteq V,|R|=k}{\operatorname{argmin}}\sum_{v\in R}d(q,v),$$

其中, $d(\cdot,\cdot)$ 表示向量空间中的距离函数, 常用的包括欧几里得距离、余弦相似度等; R 表示候选的 Top-k 查询结果集合.

召回率 (*Recall*) 是衡量向量检索质量的核心指标之一, 用于评估搜索返回结果中包含多少真实的最近邻. 设

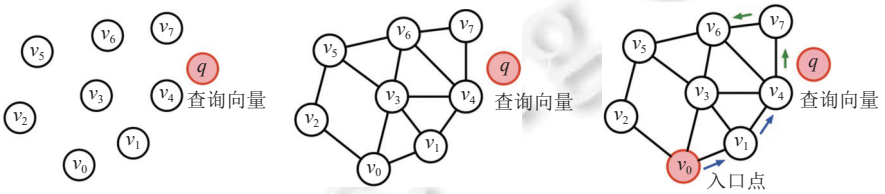
数据集中与 q 的真实最近邻集合为 $GT_k(q)$, 系统返回的近似结果集合为 $ANN_k(q)$, 则 Top-k 召回率定义为:

$$Recall@k(q)=\frac{|ANN_k(q)\cap GT_k(q)|}{k}.$$

Recall 的取值范围为 [0, 1], 数值越接近 1 表示结果越准确. ANNS 的目标是在尽可能保证召回率的前提下, 降低查询延迟与资源消耗 (如内存与 I/O 次数), 从而在大规模高维数据场景中实现可扩展的高效向量检索.

1.2 基于索引图的 ANNS

在 ANNS 中, 索引图 (index graph) 是一种在查询精度与效率之间进行了良好平衡的索引结构[16]. 其核心思想是将原始高维向量数据构建为图, 使相似向量通过边相连接, 从而在图上通过沿着边的跳转逐步逼近查询向量的近邻. 给定一个向量集合 $\mathcal{V}=\{v_1,v_2,\ldots,v_n\}\subset\mathbb{R}^d$, 其中 v_i 表示第 i 个向量, 其坐标为 $x_{v_i}\in\mathbb{R}^d$. 基于此构建近邻图 $G=(V,E)$, 其中每个顶点表示一个向量, 边集合 E 连接每个点与其若干个近邻, 构成索引图结构. 如图 2 所示, 图 2(b) 展示了由图 2(a) 中的向量数据集所构建的索引图结构.



(a) 原始向量数据 (b) 向量索引图 (c) 索引图上的搜索过程

图 2 基于索引图的 ANNS 示意图

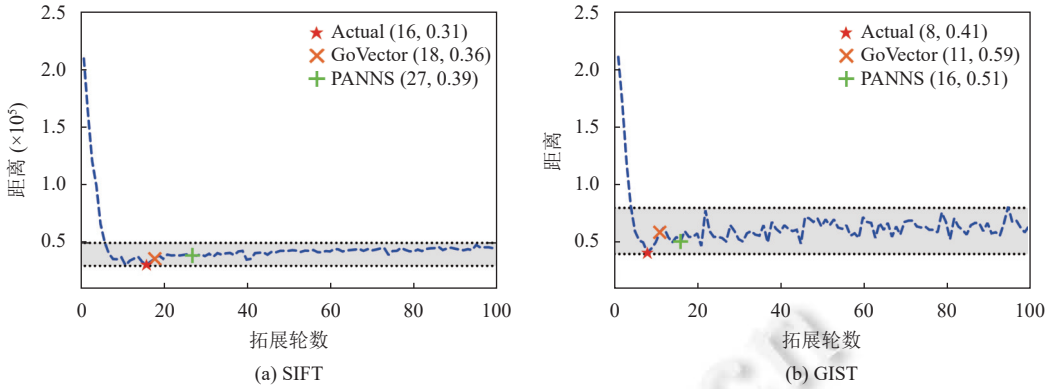
在基于索引图的近似最近邻搜索中, 针对给定的查询向量 q, 首先从图中随机选取或使用启发式策略确定一个查询入口点 v_{start}, 从该点出发, 算法将其邻居顶点按照与 q 的距离加入候选队列 L, 每轮选择候选队列 L 中与 q 最接近且未访问的点 p^* 进行拓展 (获取顶点的向量和邻居信息), 直到候选队列 L 中前 l 个顶点均被访问. 令索引图为 $G(P,E)$, 其中顶点集 P 表示数据集中的所有点, 边集 E 定义了点之间的连接关系. 对于任意点 $p\in P$, 设其向量为 x_p, 查询向量为 x_q, 距离函数为欧几里得距离 $dist(p,q)=\|x_p-x_q\|_2$. 若当前候选集合为 $L\subseteq P$, 访问集合为 $S\subseteq P$, 则每轮扩展点选择如下:

$$p^*=\underset{p\in L\setminus S}{\operatorname{argmin}}\,d(p,q),$$

同时, 更新候选队列 $L\leftarrow L\cup N_{\text{out}}(p^*)$、访问集合 $S\leftarrow S\cup\{p^*\}$, 其中, $N_{\text{out}}(p^*)$ 为 p^* 的邻居顶点. 重复上述过程, 直到 $|L\cap S|=l$, 最终返回 L 中与 q 最接近的 k ($k\leqslant l$) 个顶点作为结果. 该策略被称为 Greedy Search, 在实际实现中也常被改进为 Beam Search, 以提高并行 I/O 效率. 在图 2(c) 的示例中, 搜索过程从入口点 v_0 出发, 其 3 个邻居与查询向量 q 的距离按升序排列为 v_1,v_3,v_2, 这些点依次被加入候选集合 L. 首先选择距离最近的 v_1 进行拓展, 由于 v_3 已在 L 中, 仅需将 v_1 的另一个邻居 v_4 加入 L. 此时候选集合 $\{v_2,v_3,v_4\}$ 中, v_4 与 q 的距离最小, 故被选为下一轮拓展点. 至此, 算法已成功定位到 q 的最近邻 v_4. 为了查找 q 的 Top-k 近邻, 算法会继续按此策略拓展, 直至满足终止条件.

1.3 搜索过程的两阶段划分

ANNS 的查询过程呈现出显著的两阶段性特征[21]. 如图 2(c) 所示, 在搜索到 q 的最近邻 v_4 之后, 由于需要进一步查询剩余候选顶点以构成完整的 Top-k 返回结果, 因此还需要继续拓展搜索路径. 此时, 拓展路径可能逐渐远离查询点 q 导致距离整体呈现上升趋势, 但由于搜索仍局限在局部候选区域的若干跳范围内, 且扩展仍倾向于优先探索距离较近的顶点, 因此距离值通常只会在一个较小的区间内波动. 图 3 展示了在 100 万个向量的 SIFT 和 GIST 数据集上进行 ANNS 查询时, 查询点 q 与每轮拓展点 p^* 之间的欧氏距离随着拓展轮数的变化趋势. 图中的红点标注了查询过程中最接近目标向量的顶点出现的拓展轮数及对应的最小距离. 可以观察到, 在到达这一最小距离点之前, 查询距离呈现快速下降趋势; 而在此之后, 距离变化逐步进入较为稳定的波动区间. 基于这一结果, 本文在分析 ANNS 的查询过程中使用两阶段的概念.



(a) SIFT (b) GIST

图 3 不同数据集上 ANNS 查询过程中拓展点 p^* 与查询点 q 的距离变化 (k=10)

第 1 阶段为快速接近查询向量阶段. 多数基于图结构或树结构的 ANNS 算法 (如 HNSW、NSG、KD-Tree[22]) 通常从一组入口顶点开始搜索. 尽管这些入口顶点可能距离查询向量较远, 算法仍可通过图跳跃或树的快速分支遍历, 在较少跳数内迅速接近查询向量, 此时查询距离迅速下降. 第 2 阶段为 Top-k 结果的精细探索阶段. 此时, 算法已定位到局部最近邻区域, 但为了获得最终的 Top-k 最优解, 仍需扩展多个邻域分支以排除潜在更优的候选. 该阶段搜索范围更广, 路径呈现较高的不确定性, 并可能访问到一些距离查询向量更远的候选点, 导致查询距离出现波动甚至上升的现象.

尽管现有研究已通过向量压缩、路由预测等技术对第 1 阶段进行了优化[16,23,24], 但在第 2 阶段, 由于候选扩展频繁、访问模式分散且数据局部性较差, 传统缓存策略难以有效发挥作用, 频繁的磁盘 I/O 成为制约查询性能的主要瓶颈. 因此, 本文聚焦于优化查询的第 2 阶段, 通过提升数据局部性与缓存命中率, 有效减少查询过程中的磁盘访问次数, 提升整体检索效率.

2 研究背景与相关工作

2.1 向量索引

在高维语义向量广泛应用于自然语言处理、推荐系统和图神经网络[25]等任务的背景下, 如何设计高效的向量索引结构, 以支撑大规模、高并发、低延迟的 ANNS, 已成为向量数据库系统的核心挑战. 传统精确检索方式在高维空间中面临“维度灾难”, 索引构建与查询效率急剧下降, 难以满足在线服务场景的性能需求. 为在检索效率与精度之间取得平衡, ANNS 索引技术应运而生, 并逐步演化为工业级向量数据库系统 (如 Faiss[26]和 Milvus[18]) 的基础组件. 现有主流的向量索引方法大致可分为如下 3 类.

(1) 哈希索引 (如 LSH[27,28]): 通过构造多个哈希函数将向量映射至不同的桶 (bucket), 仅在相同桶内执行候选搜索. 该方法计算开销低, 但在追求高召回率时需构建大量哈希表, 导致内存占用与系统维护成本增加, 扩展性有限.

(2) 量化索引 (如 IVF_PQ[24]、SCANN[29]): 对向量空间进行压缩或分区, 以降低计算与存储成本. 此类方法在工业系统中广泛应用, 如 Faiss 使用乘积量化加速候选聚类定位. 然而, 在高精度检索任务中, 量化误差可能导致相似向量被映射至不同单元, 降低召回率与整体检索效果.

(3) 索引图 (如 HNSW、NSG、DiskANN): 通过构建稀疏可导航的近邻图, 并结合启发式遍历策略, 实现高维向量空间中的高效近似搜索. 索引图方法近年来凭借优异的精度与可扩展性, 逐渐成为处理大规模向量检索任务的主流方案.

2.2 基于索引图的 ANNS 优化技术

索引图方法因其高准确率和良好扩展性受到广泛关注. 其核心思想是: 在构建索引时预计算部分近邻关系, 将向量作为顶点, 相似度为边, 构建稀疏有向图结构. 查询过程中, 从入口顶点出发, 通过近邻扩展算法实现快速定位

目标近邻. 在典型的索引图方法中, HNSW 通过多层跳跃图结构实现高效的内存搜索; NSG 通过约束图结构的稀疏性, 优化搜索路径长度和冗余边数; DiskANN 提出了一种面向磁盘的索引图结构, 通过 Beam Search 搜索算法支持大规模向量检索, 已被集成至 Milvus 等工业系统.

(1) 存储优化技术

存储布局是基于磁盘的索引图系统的核心设计要素, 其物理组织方式直接决定了检索过程中的磁盘 I/O 效率. 当前主流向量数据库系统主要采用两种基础存储策略: 插入顺序存储与哈希分布存储. 插入顺序存储按照向量插入的时序线性组织数据, 虽然实现简单且写入吞吐量高, 但由于完全忽略向量的相似性, 导致高维空间中相邻的向量在物理存储上呈现高度离散分布, 使得查询时需要触发大量随机 I/O 操作. 哈希分布存储通过哈希函数实现数据的均匀分布, 能够有效避免访问热点问题, 但同样割裂了图结构中邻接顶点的物理存储位置, 导致跨页访问现象加剧.

针对上述问题, 学界提出了多种创新解决方案. 例如, Facebook 的 Faiss-IVFOPQ[23]通过倒排索引 (inverted file, IVF) 聚类相似向量, 并配合乘积量化 (PQ) 压缩存储, 同时减少了磁盘寻道次数和单次 I/O 数据传输量. UC Berkeley 提出的 LENS 系统基于查询日志挖掘访问热点, 进行主动预加载, 实验结果表明可将缓存命中率提升约 25%[30].

(2) 缓存优化技术

为了降低查询过程中的磁盘访问延迟, 现代 ANNS 系统普遍引入缓存机制, 以缓解因图结构跳转不确定性导致的频繁 I/O 问题. 缓存机制主要通过在查询路径上提前加载部分关键顶点数据, 从而提升查询阶段的数据命中率与响应效率. DiskANN 等方法通过提前加载访问频率较高的入口顶点及其若干跳邻居, 来减少从磁盘加载数据的次数. Starling 系统则引入了内存导航图的概念, 依据系统的内存限制, 从整个数据集中随机采样一部分顶点, 并利用构建算法 (如 Vamana) 在内存中构建轻量级导航图. 该导航图能够高效地为查询向量提供更接近的入口点, 从而显著缩短其在磁盘图上的搜索路径, 有效降低 I/O 负担.

3 方法概览

本文提出 GoVector, 一种基于向量相似性的 I/O 高效向量近邻查询缓存策略. 其方法整体框架如图 4 所示.

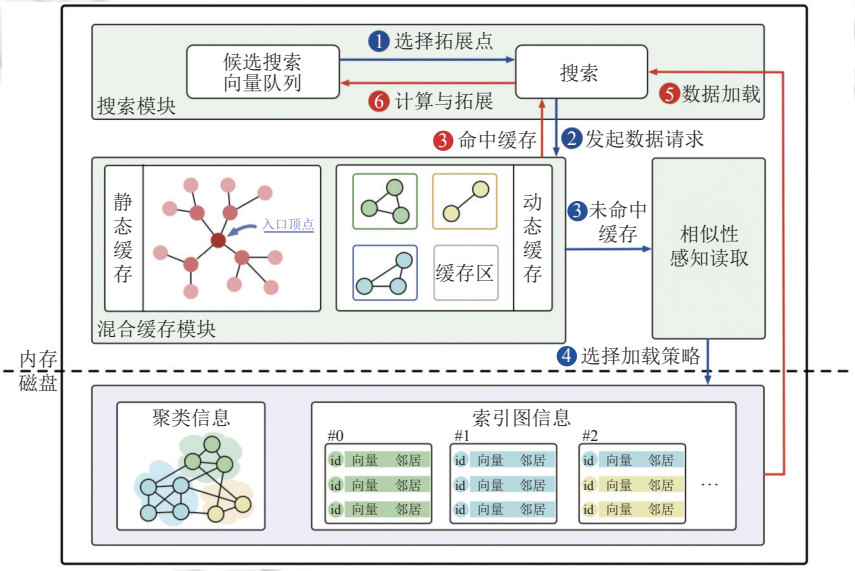


图 4 GoVector 系统架构图

在内存层面, GoVector 设计了一种静态与动态结合的混合缓存机制, 以适应 ANNS 查询过程中不同阶段的访问行为. 在静态缓存区, 系统根据预设的容量, 从入口点出发预加载其多跳邻居, 用于快速定位到查询向量附近的

候选区域. 在动态缓存区, 系统自适应地缓存当前查询过程中访问到的顶点所在的页面及其在磁盘中邻近的页面, 用于支持 Top-k 相似向量的扩展访问. 在磁盘层面, GoVector 首先基于向量相似性对查询点邻近区域内的向量进行聚类, 并据此对原始索引图中的向量进行重排序. 重排序后的向量再依据分区存储至磁盘, 以提升数据访问的局部性, 减少跨页加载频次.

图 4 展示了该搜索流程的主要步骤. ① 从候选向量队列中选取当前距离最近且尚未访问的顶点进行拓展. ② 根据拓展点的 ID 向混合缓存模块发起数据读取请求. ③ 混合缓存模块判断该顶点是否已驻留于缓存中, 若命中则返回其向量和邻居信息, 否则向读取模块发起磁盘访问请求. ④ 读取模块根据当前所处的搜索阶段来选择不同的磁盘加载策略, 若当前处于搜索的第 1 阶段, 则直接读取拓展顶点所在的磁盘页, 反之则采用相似性感知读取机制, 同时加载拓展点及其相似向量所在的多个页. ⑤ 将读取到的顶点向量和邻居信息加载至内存, 若当前阶段为第 2 阶段, 还需将相关页面写入动态缓存. ⑥ 计算拓展点与查询向量之间的真实距离, 并依据 PQ 距离将其邻居加入候选向量队列中.

4 静态-动态混合缓存机制

4.1 现有缓存策略局限性分析

传统的 Beam Search 搜索算法通常采用静态缓存机制, 即在系统初始化阶段, 根据预设的缓存容量, 预加载入口顶点及其若干跳邻居, 并在整个搜索过程中保持缓存内容不变. 该机制在搜索的初始阶段具有较高的缓存命中率, 因为每轮查询均从入口顶点出发, 路径前几跳的顶点被频繁访问, 因而能够有效提升命中率. 然而, 静态缓存机制无法感知查询路径的实际拓展行为, 导致其命中范围往往局限于搜索路径的初始几跳. 随着搜索的深入, 命中率迅速下降, 导致后续阶段的访存效率大幅降低. 我们基于 DiskANN 在 SIFT 与 GIST 两个数据集上进行了实验测试, 结果如图 5 所示. 随着查询拓展轮数的增加, 静态缓存的命中率呈现明显的幂律衰减趋势. 结合第 1.3 节中的两阶段搜索划分标准, 我们将找到精确最近邻的轮次定义为两个阶段的转折点, 并在图 5 进行标注. 可以观察到, 当搜索进入第 2 阶段后, 静态缓存的命中率相比于第 1 阶段显著下降. 在第 1 阶段, 两个数据集的静态缓存命中率分别为 19% 和 63%; 而在第 2 阶段, 该命中率大幅下降, 仅为 4% 和 9%. 此外, 实验结果还表明, 第 2 阶段的搜索耗时通常占据总搜索时间的 80% 以上, 说明该阶段已成为制约整体检索效率的主要性能瓶颈.

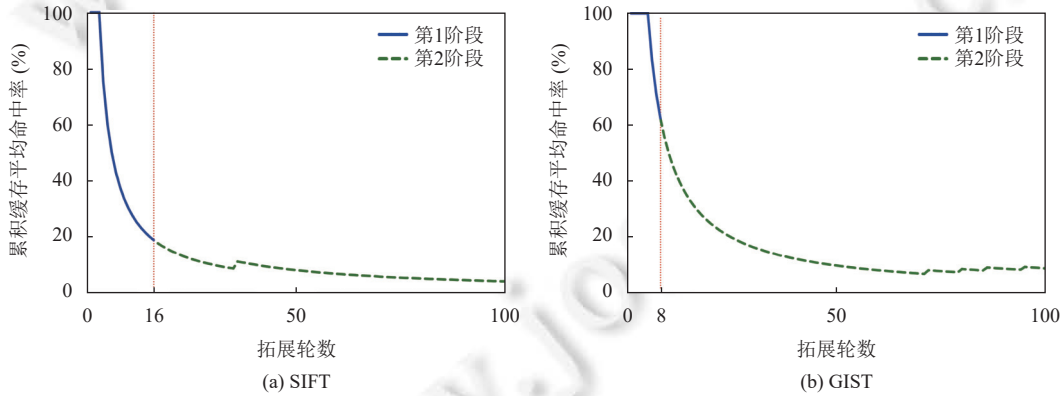
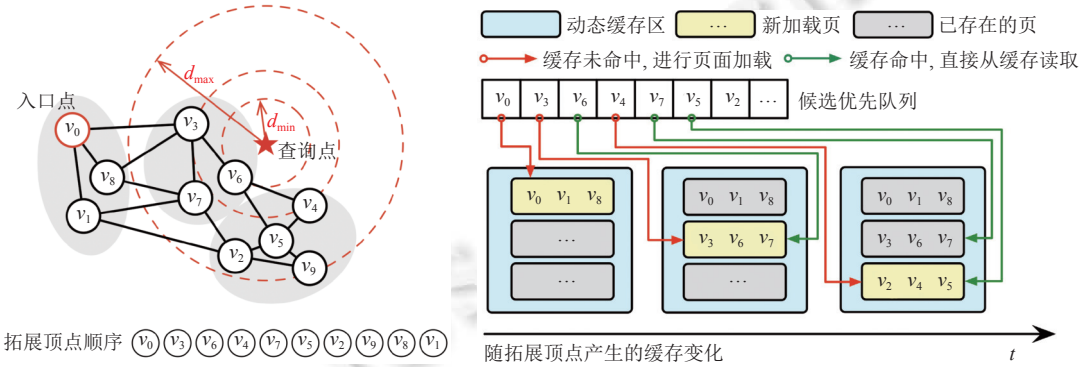


图 5 DiskANN 100 轮拓展中静态缓存命中率变化曲线 (k=10)

为了更深入地理解第 2 阶段的拓展特性, 我们进一步分析了该阶段拓展的顶点与查询向量之间的距离分布. 图 3 展示的实验结果表明, 在该阶段中, 拓展的顶点 p^* 在向量空间中与查询向量 q 的距离集中落在一个窄幅区间内, 波动较小, 表现出显著的空间聚集性. 设 $d(p^*,q)$ 表示拓展顶点与查询向量之间的距离. 根据图 3 的实验结果, 可得 $d_{\min} < d(p^*,q) < d_{\max}$. 其中, $d_{\min}$ 和 $d_{\max}$ 分别表示该阶段访问点与查询向量之间的最小与最大距离. 由此可见, 第 2 阶段的查询过程实际上被限制在一个以 q 为中心、半径范围为 $[d_{\min}, d_{\max}]$ 的环形区域内进行扩展. 相较于整

个向量空间, 该局部区域内的向量在空间上更为接近, 也更有可能被访问以用于距离计算和候选集扩展. 图 6(a) 展示了该环形区域的结构示意图. 基于这一观察, 如果第 2 阶段的查询过程能够聚焦于该区域内的向量, 对其进行有序组织并采用对应的物理存储布局, 使缓存的换入与换出操作局限于该区域所覆盖的磁盘页之间, 则每次磁盘 I/O 所加载页面中的向量将具有更高的实际访问概率, 从而显著提升缓存命中率, 减少冗余 I/O 开销. 然而, 当前主流的基于磁盘的索引图方法 (如 DiskANN) 在搜索过程中仍采用逐点加载策略, 仅加载当前拓展顶点所在的磁盘页, 并在获取该顶点的向量与邻居信息后立即将其从内存中淘汰, 忽略了第 2 阶段中拓展顶点在向量空间中的聚集性特征, 未能有效利用访问路径中的空间局部性, 从而可能引发重复的磁盘 I/O 操作. 因此, 第 1 阶段的查询过程仍可采用传统静态缓存机制, 而第 2 阶段则应对现有策略加以改进, 设计聚焦于局部区域的缓存机制, 以更高效地支持查询路径中频繁访问的热点区域.



(a) 第2阶段搜索向量扩展的范围　　(b) 顶点拓展过程和缓存的动态更新

图 6　动态缓存示意图

4.2　查询感知的混合缓存设计

基于第 4.1 节的分析结果, 本文提出了一种静态-动态结合的混合缓存机制, 即搜索的第 1 阶段采用静态缓存策略 (与传统的 Beam Search 一致), 用于加速起始路径的高频访问; 当搜索进入第 2 阶段后, 系统启用动态缓存机制, 重点优化访问局部性较强的路径. 具体而言, 在第 2 阶段的搜索过程中, 对于缓存未命中的拓展顶点, GoVector 会结合该顶点在向量空间中的位置, 触发一次批量读取操作, 将其邻近区域内的多个向量一并从磁盘加载至动态缓存中. 这种策略旨在充分利用拓展点之间的空间相似性, 使后续拓展的顶点更有可能命中缓存, 从而显著减少频繁的随机 I/O 操作所带来的性能开销. 如图 6(b) 所示, 当系统拓展顶点 v_3 时, 由于缓存未命中, 系统根据其在向量空间中的邻近性, 一次性地将 v_3、v_6 和 v_7 所在的缓存页加载至动态缓存区. 由于候选优先队列中的顶点具有较高的拓展概率, 系统在后续拓展过程中能有效复用之前加载的缓存页. 例如, 当拓展到顶点 v_6 和 v_7 时, 其所对应的缓存页已由此前拓展 v_3 时加载入缓存, 因而可直接命中, 避免重复的磁盘访问, 从而显著提升整体查询效率.

为了适配该动态缓存机制, GoVector 设计了一种相似性感知读取策略. 为了支持顺序批量加载, GoVector 在索引构建阶段对向量数据进行了空间布局优化, 使得相似向量能够按类进行划分与存储, 从而在物理层面上实现良好的局部性. 因此, 在执行读取操作时, 系统能够以顺序访问的方式定位相关页面, 比传统的随机 I/O 开销更低. 该相似性感知读取机制的关键在于, 针对每个待拓展的目标顶点, 系统动态确定其所属类及其类内位置, 并结合当前查询上下文, 计算出最合适的顺序读取区间, 实现局部性增强的批量加载操作. 具体来说, 每次拓展目标点时, 系统首先识别目标点所属的类 (目标类), 再结合类的大小与拓展点在其中的相对位置, 计算最优的顺序读取起止区间, 实现更具局部性的批量加载行为. 自适应数据读取机制遵循以下 3 条原则: ① 以目标点为中心, 优先加载其所属类中的相邻顶点; ② 在当前类无法满足读取需求时, 优先从目标类及其相邻类中补充加载顶点; ③ 保证读取范围不发生越界.

图 7 展示了实际应用中常见的 3 种典型场景, 假设缓存页大小为 4. 在情况 1 中, 目标类的大小不小于缓存页

容量, 因此以 v_5 为中心, 系统可直接从同一类中加载相邻的 v_2、v_5、v_9、v_4 这 4 个顶点, 完成一次有效缓存填充; 在情况 2 中, 由于目标类 (类 3) 较小, 无法独立填满缓存页, 系统将以类 2 为中心, 同时补充加载类 2 中与目标点相邻的顶点 (v_4); 在情况 3 中, 若直接按照上述原则读取会导致磁盘读取范围的左边界超出索引文件, 因此系统会执行边界检测与调整机制, 以避免读取异常, 确保读取行为的正确性.

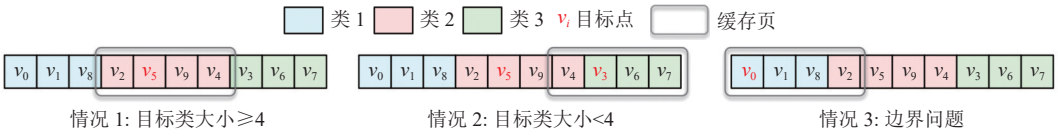


图 7 相似性感知读取机制示意图

在静态缓存机制的基础上, GoVector 首次引入了查询感知的动态缓存区. 但由于在向量搜索的过程中, 动态缓存区的数据量会随着搜索路径的扩展不断增长, 当其达到设定的容量上限时, 系统需执行缓存替换操作. 为此, 本文借鉴操作系统中虚拟内存管理的思想, 实现了 3 种替换策略: 先进先出 (first-in-first-out, FIFO)、随机替换 (Random) 以及最不经常使用 (least frequently used, LFU) 策略. 通过对 3 种策略的对比评估, 最终选择 LFU 作为动态缓存的默认替换策略.

我们针对搜索的两阶段提出了不同的缓存方式, 因此, 如何准确识别两个阶段的转折点成为实现高效搜索的关键. 本文借鉴了 PANNS[21]中提到的识别方法, 即当候选队列中排名前 k 的顶点均已被访问时, 判定搜索进入第 2 阶段. 然而, 在实际应用中, 该识别策略存在明显的滞后性, 即转折点往往在真实转折发生后经过多轮拓展才被检测到. 如图 3 所示, 在 SIFT 和 GIST 数据集中, 距离查询点最近的向量分别出现在第 16 轮和第 8 轮, 即第 1、2 阶段的真实转折点, 而 PANNS 识别的转折点分别出现在第 27 轮和第 16 轮. 为了克服转折点检测滞后这一问题, 本文在此基础上引入了一个判断参数 θ ($0<\theta<1$), 即当候选队列中排名前 $\theta \cdot k$ 的顶点均已被访问时, 判定搜索进入第 2 阶段. θ 通过如下方法确定: 从数据集中随机抽取 1% 的查询向量样本, 记录每个查询在完整搜索过程中的真实转折轮数 k (即首次访问 Top-k 最近邻所需的轮数), 并结合 PANNS 方法计算其估计转折轮数 k'. 最终以 $\theta = k/k'$ 的方式获得各查询的转折判断参数, 用于动态识别搜索阶段, 从而实现阶段切换的控制. 这种方法能够更早感知搜索阶段的变化, 显著缓解转折点识别的滞后问题. 在图 3 的实验中, 相较于 PANNS, GoVector 在 SIFT 和 GIST 数据集中识别到的转折点分别提前 9 轮和 5 轮, 验证了该方法的有效性.

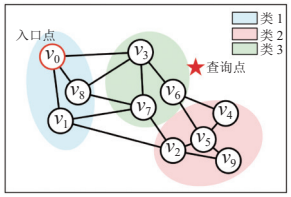
动态缓存机制对 ANNS 查询性能的优化效果, 不仅依赖于缓存策略的设计, 还与底层存储布局的局部性特征密切相关. 若相似向量能够聚集于同一磁盘页中, 则顺序读取过程中加载的向量更有可能被访问, 从而进一步提升缓存的命中率. 因此, 优化存储布局以提升 I/O 的访问局部性, 成为缓存机制之外的另一关键问题. 为此, 第 5 节将重点探讨基于向量相似性的索引图重排序方法.

5 基于向量相似性的索引图重排序

5.1 I/O 效率问题分析

在大规模向量检索任务中, 磁盘 I/O 成本往往成为限制系统性能的关键瓶颈, 其另一原因在于单次 I/O 的有效利用率偏低. 由于磁盘以页为基本读写单位, 系统在访问某个顶点的特征或邻居信息时, 必须整体加载该顶点所在页的全部内容. 若该页面中仅有少数顶点实际参与查询过程, 其余数据则为无效读取, 造成带宽浪费. 为了继续搜索其他相关向量, 系统不得不频繁加载新的磁盘页, 从而引发额外的磁盘 I/O 开销. 已有研究指出[16], 在 DiskANN 系统中, 每次读取的数据页中约有 94% 的顶点未被访问, 查询过程有超过 92.5% 的时间用于磁盘 I/O. 该问题在 ANNS 查询的第 2 阶段尤为突出, 此阶段虽已定位到部分与查询向量接近的初始候选, 但为了进一步优化结果, 还需在其邻近的向量空间中进行更深入的拓展. 若能将当前拓展顶点与其近邻的其他向量集中存储于同一磁盘页内, 则系统一次 I/O 操作就能加载多个有可能被命中的向量, 有效提升单次 I/O 的利用率, 减少冗余磁盘访问. 然而, 目前主流的数据布局策略仍主要基于图结构连接关系组织页面, 如 Starling 系统采用 BNF (block neighbor

frequency) 算法, 试图将图中相邻顶点聚集在同一块中. 这种方法忽视了图结构与向量空间之间的结构差异, 无法充分保证相似向量的物理邻近性, 进而限制了 I/O 局部性的进一步提升. 图 8 展示了不同方法的存储布局与缓存命中情况 (假设没有静态缓存). 图 8(a) 为索引图结构; 图 8(b) 展示了 DiskANN、Starling 与 GoVector 的存储布局结果, 均假设每页最多容纳 3 个顶点; 图 8(c) 模拟了从入口点开始进行搜索的拓展过程, 并记录了每一轮拓展顶点、候选优先队列, 以及当前页面是否命中下一个拓展顶点. 实验过程中, 从入口点 v_0 出发, 系统依次拓展点 v_0, v_3, v_6, 并维护候选队列. 由于 v_6 距离查询点最近, 因此第 3 轮拓展 v_6 后进入第 2 搜索阶段, 此时的候选队列为 $\{v_4, v_7, v_5, v_8, v_1\}$, 下一拓展点为 v_4. 若采用 DiskANN 的存储策略, 由于 v_6 与 v_4 存储于不同磁盘页, 会产生两次磁盘 I/O 操作; 在 Starling 的布局下, 二者分布于不同页面, 则仍需两次磁盘 I/O 操作. 直至搜索结束, 两种存储布局均因划分不合理, 导致单次 I/O 的数据利用率低下. 尽管可以通过扩大缓存容量或引入异步加载机制来缓解 I/O 开销, 但如果能设计出更合理的索引布局方案, 将以更低成本取得更显著的性能提升.



(a) 索引图结构

索引页面存储布局

DiskANN	Starling	GoVector
v_0, v_1, v_2	v_0, v_3, v_8	v_0, v_1, v_8
v_3, v_4, v_5	v_1, v_2, v_7	v_2, v_5, v_9
v_6, v_7, v_8	v_4, v_5, v_6	v_4, v_3, v_6
v_9	v_9	v_7

(b) 不同方法的存储布局

拓展顶点	候选队列	下一个拓展点是否命中当前页		
		DiskANN	Starling	GoVector
v_0	v_3, v_8, v_1	✘	✔	✘
v_3	v_6, v_7, v_8, v_1	✘	✘	✔
v_6	v_4, v_7, v_5, v_8, v_1	✘	✔	✔
v_4	v_7, v_5, v_8, v_1	✘	✘	✘
v_7	v_5, v_2, v_8, v_1	✘	✘	✘
v_5	v_2, v_9, v_8, v_1	✘	✘	✔
v_2	v_9, v_8, v_1	✘	✘	✔
v_9	v_8, v_1	✘	✘	✘
v_8	v_1	✘	✘	✔

(c) 不同方法缓存命中情况

图 8 基于向量相似性的索引图重排序

5.2 基于向量相似性的索引图重排序方法

针对现有布局方法存在的局限性, 本文提出了一种结合向量相似性与存储重排的索引图布局优化策略, 以提升磁盘访问效率与整体检索性能. 具体来说, GoVector 通过以下两个阶段实现索引图的重排序.

(1) 相似性聚类阶段: 基于向量空间中的欧氏距离, 采用 K-means 聚类算法[31,32]将全部向量划分为多个高相似度簇, 每个簇内的向量在向量空间中彼此接近.

(2) 局部性优化阶段: 在聚类结果基础上, 结合索引图的拓扑结构, 将同一簇内的向量尽可能分配至相同或物理相邻的磁盘页中, 以最大程度地降低查询过程中的跨页访问频率.

图 8 展示了该重排序方法的过程与效果. 首先, 系统在原始向量空间中应用 K-means 聚类算法, 将向量划分为类 1–类 3, 对应的索引图结构如图 8(a) 所示. 随后, 在每个聚类内部, 系统进一步基于图拓扑中顶点之间的连通性进行顺序重排. 以类 3 为例, 其中包含的顶点集合为 $\{v_2, v_4, v_5, v_9\}$, 但由于磁盘页面大小存在限制, 系统在排序时优先将 $\{v_2, v_5, v_9\}$ 顺序组织在同一页面中, 因为这 3 个点相对于 v_4 更接近所在类的质心, 而边缘的顶点 v_4 则被安排在下一个页面中, 最终重排后的布局结果如图 8(b) 中 GoVector 所示. 在实际查询过程中, GoVector 能够有效提升单次 I/O 的数据利用率. 例如, 以传统 Beam Search 算法为例, 当拓展顶点 v_3 时, 系统会加载 v_3 所在的磁盘页. 然后继续拓展顶点 v_6, 由于 v_6 与 v_3 在 GoVector 存储布局下处于同一磁盘页, 因此可以减少一次磁盘 I/O, 充分利用了上一次的 I/O 带宽. 图 8(c) 对比了不同存储布局策略下的实际缓存命中情况. DiskANN 使用向量插入顺序或随机分布方式进行磁盘存储, 完全忽略向量之间的相似性关系, 导致在查询过程中, 相似向量往往被分散在不同磁盘页中. 在本例中, 由于下一个拓展点都无法命中当前页, 因此需要执行更多次 I/O 操作, 造成极高的 I/O 开销. Starling 采用图拓扑结构作为分布依据, 将拓扑邻居尽可能聚集于同一磁盘页中, 能够在一定程度上缓解跨页问题. 但在 ANNS 的第 2 阶段, 拓展路径是由向量与查询点之间的相似性驱动, 而不是由拓扑结构决定的. 由于图结构与相似性结构之间并不总是一致, Starling 的布局方式难以精准覆盖实际访问路径中那些具有空间聚集性的热点区域. GoVector 首先通过聚类方法识别出向量空间中高相似度的局部区域, 然后在每个聚类内结合图拓扑结构进行局

部排序与页分配, 使得在实际查询过程中, 相似向量更可能被一并加载至同一磁盘页中. 这种布局更贴合第 2 阶段以相似性为导向的拓展行为, 因此, GoVector 的布局方式有效提升了单次 I/O 的数据利用率, 使得下一次拓展顶点与当前拓展顶点更大概率位于同一磁盘页中, 从而减少 I/O 开销并提升查询性能.

6 实验分析

6.1 实验设置

6.1.1 实验环境

本文所有实验均在一台高性能服务器上完成, 该服务器配备 Intel® Xeon® Gold 6248R 处理器 (3.00 GHz, 48 核心), 32 GB DDR4 内存 (3 200 MT/s) 以及两块 1.7 TB SSD, 顺序读写带宽最高可达 500 MB/s. 操作系统为 Ubuntu 22.04 LTS, 编译器版本为 GCC 11.4.0.

6.1.2 实验数据

实验采用 6 个公开的真实向量数据集 (见表 1), 包括 SIFT[33]、Text2Img[34]、DEEP[8]、Word2Vec[8]、MSong[8] 和 GIST[33]. 这些数据集涵盖图像、文本、音频和词向量等多种类型, 向量维度从 128 至 960 不等, 已被广泛用于现有 ANNS 系统的性能评估.

表 1 实验数据集

数据集	数据集类型	向量维度	向量数量	查询数量	内容类型
SIFT	float	128	1 000 000	10 000	图像
Text2Img	float	200	1 000 000	1 000	图文混合
DEEP	float	256	1 000 000	1 000	图像
Word2Vec	float	300	1 000 000	1 000	词向量
MSong	float	420	994 185	1 000	音频
GIST	float	960	1 000 000	1 000	图像

6.1.3 对比系统及参数设置

本文将 GoVector 与当前两种代表性的基于磁盘的近似最近邻搜索系统 DiskANN 和 Starling 进行对比.

DiskANN 是由微软开发的一种基于图的高效磁盘 ANNS 方法, 采用贪婪搜索策略, 从预选的入口顶点出发, 沿图的邻接结构逐步逼近查询向量. 为了减少 I/O 开销, DiskANN 在内存中引入了静态缓存机制, 预加载访问频率较高的入口顶点及其若干跳邻居. 本文中对其实验参数设置如下: 缓存顶点数为索引文件总大小的 1%, 默认执行 Top-100 查询, 图中每个顶点的邻居数 R 为 32, 搜索线程数 T 设为 32.

Starling 是近年来提出的一种磁盘驻留型索引图系统, 针对分段式数据布局优化了存储与查询路径. 其查询过程中采用块级搜索策略, 以磁盘页为单位加载数据, 有效降低路径长度与 I/O 频次. 此外, Starling 基于拓扑结构引入重排序算法 (BNF 策略), 使得相邻顶点尽量聚集于同一磁盘页中, 从而提升 I/O 利用率. 在本文实验中默认重排序策略为 BNF, 其余参数设置与 DiskANN 保持一致.

GoVector 是本文提出的一种高效的混合缓存策略. 它通过静态与动态缓存相结合的机制, 在不增加内存占用的前提下提升 I/O 命中率与整体查询吞吐量. GoVector 的核心思想是基于向量的相似性对索引进行物理布局优化, 使在查询过程中拓展路径上的相邻顶点尽可能集中在同一磁盘页中, 从而提升局部性 (区别于 Starling 的拓扑相似性重排序). 实验中设置的参数为: 静态缓存与动态缓存在总缓存顶点数中的比例设为 2:8, 其余参数与 DiskANN 一致.

6.2 系统整体表现

图 9 展示了不同 ANNS 方法在每秒处理的查询次数 (queries per second, QPS) 与召回率 (*Recall*) 之间的性能对比. 其中, GoVector-hybrid 表示采用静态与动态缓存机制相结合的 GoVector (静态与动态缓存比例为 2:8), GoVector-dynamic 则表示未使用静态缓存 (即静态缓存顶点数为 0) 的纯动态版本 GoVector.

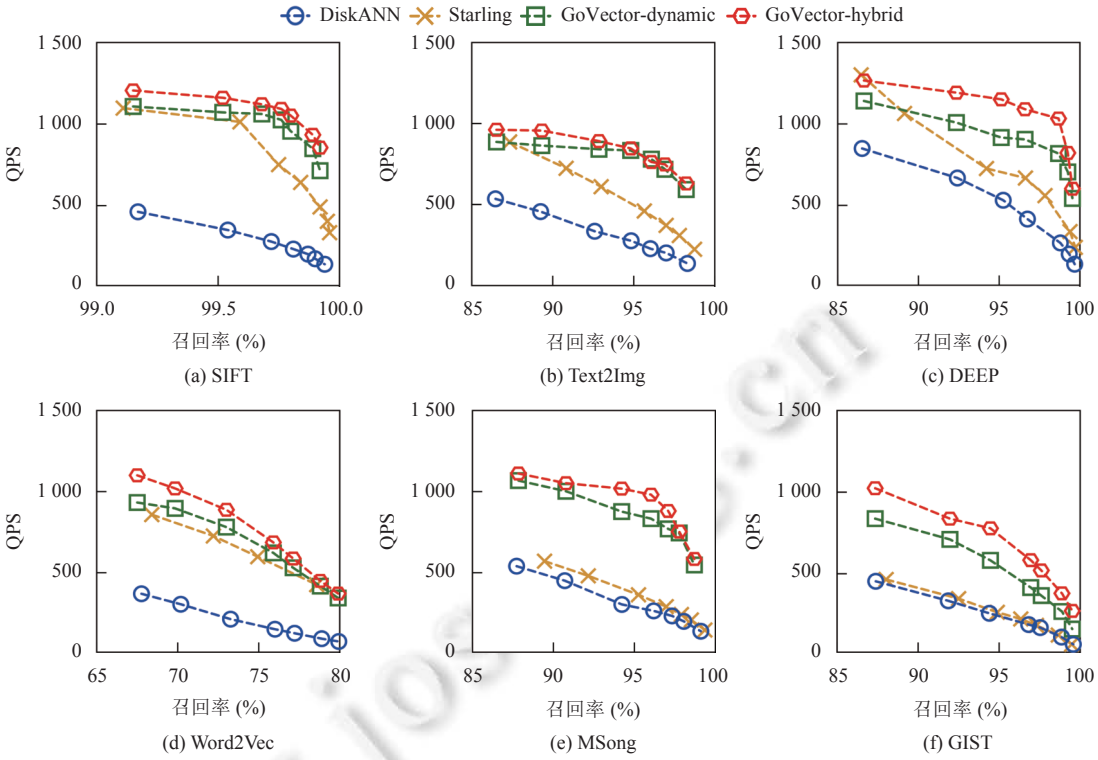


图 9 不同 ANNS 方法的召回率-QPS 性能对比

从实验结果可以看出, GoVector 在与其他 ANNS 方法的对比中展现出优异的搜索性能, 尤其在高召回率场景下优势更为明显. 具体而言, 在召回率不低于 90% 的条件下, GoVector-hybrid 的 QPS 相比于 DiskANN 提升了 2.61–4.59 倍, 相比于 Starling 提升了 1.10–3.97 倍, 相比于 GoVector-dynamic 提升了 1.06–1.50 倍. 进一步观察可知, 随着数据维度的提升, GoVector 依然保持了较高的搜索效率, 而 Starling 的性能接近于 DiskANN, 这得益于 GoVector 的高效混合缓存策略以及基于向量相似性的布局优化.

值得注意的是, 在少量低召回率场景 (如 Text2Img 和 DEEP 数据集) 中, Starling 的 QPS 略优于 GoVector. 这是因为在该类场景下, 搜索的第 1 阶段占据了大部分耗时, 而 GoVector 所引入的动态缓存机制主要优化第 2 阶段的访问效率, 因此其优势尚未充分发挥.

此外我们还发现, 在低召回率场景下, GoVector-dynamic 的搜索性能明显低于 GoVector-hybrid, 主要原因在于 GoVector-hybrid 在第 1 阶段借助静态缓存预加载了入口顶点的多跳邻居, 有效提高了命中率, 显著减少了磁盘 I/O 访问次数, 从而提升了整体性能. 而在高召回率场景下, 随着搜索队列的拓展, 第 2 阶段成为主要瓶颈, 此时动态缓存机制发挥主导作用, 因此 GoVector-dynamic 和 GoVector-hybrid 在该场景下表现接近.

6.3 静态动态缓存比例的分析

本节在 SIFT 和 GIST 两个公开数据集上评估了不同静态与动态缓存比例对搜索性能的影响, 针对多个召回率目标进行了实验. 相关实验结果如图 10 所示. 基于实验数据分析, 可得出以下 3 个方面的结论.

① 合理的缓存配置对查询性能的提升. 实验结果表明, 当静态与动态缓存的比例为 2:8 (即动态缓存占比为 80%) 时, 在不同召回率下均能获得最优的搜索性能. 在 GoVector 中, 静态缓存主要负责加速第 1 阶段的搜索, 而动态缓存主要优化第 2 阶段的数据访问. 当动态缓存占比超过 80% 时, 静态缓存所占容量的减少使其无法充分缓存入口点及其多跳高频邻居, 导致搜索初期难以及时命中相关数据, 第 1 阶段的查询耗时上升. 尽管动态缓存对第 2 阶段依然有效, 但整体查询流程中第 1 阶段效率下降从而拉低了整体的 QPS. 这表明, 采用合理的缓存配置可

以在性能与资源利用之间达到良好的平衡.

② 动态缓存的性能提升作用. 在相同召回率条件下, 引入少量动态缓存即可显著提升系统的整体搜索性能. 例如, 在 SIFT 数据集中, 动态缓存占比从 0% 提升至 10% 时, 系统性能提升了 1.31–5.04 倍. 这充分说明了动态缓存机制在识别访问热点、适应搜索路径变化方面的高效性.

③ 动态缓存的稳定性与鲁棒性. 进一步分析发现, 在不同召回率水平下, 动态缓存的性能变化趋势保持一致, 表现出良好的稳定性. 此外, 在任意非零动态缓存占比的设置下, 系统的搜索性能均优于完全静态缓存 (即动态缓存占比为 0%), 这说明动态缓存机制在多种检索精度需求下都具备较强的通用性和适应性.

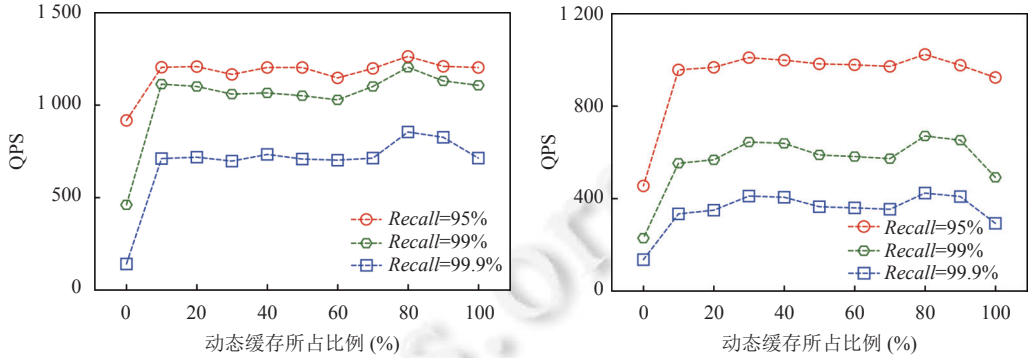


图 10 不同静态动态缓存比例下的搜索性能

6.4 不同缓存替换策略的影响分析

为了探索适用于动态缓存的最优替换策略, 本文在 SIFT 数据集上分别评估了 3 种常见策略的性能表现: 最不经常使用 (LFU)、先进先出 (FIFO) 和随机替换 (Random). 图 11(a) 展示了在不同召回率设定下, 3 种替换策略对应的搜索性能表现. 实验结果表明, LFU 策略在整体性能上表现最佳.

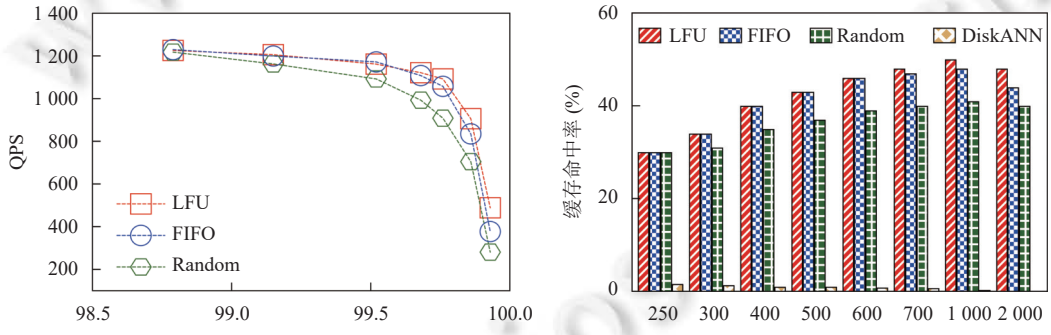


图 11 不同缓存替换策略的表现

图 11(b) 进一步展示了在不同优先队列长度下各替换策略的缓存命中率. 首先, 随着优先队列长度的增加, GoVector 动态缓存的命中率始终显著优于 DiskANN 所采用的静态缓存策略. 这一现象说明, 动态缓存机制能够更有效地适应查询过程中的访问局部性变化, 及时捕捉潜在热顶点, 从而显著减少磁盘 I/O 操作, 提升整体搜索性能. 相比之下, DiskANN 使用的静态缓存为固定顶点集合, 无法随查询路径的动态变化进行内容调整, 限制了其缓存命中率和系统效率的进一步提升. 进一步分析表明, 当优先队列长度为 250 时, 3 种动态缓存替换策略的缓存命中率基本一致, 且搜索性能差距较小. 原因是此时缓存尚未触发替换操作, 缓存内容仍处于初始化状态. 当优先队列长度在

300–600 之间时, LFU 与 FIFO 策略的缓存命中率基本持平. 这是由于 LFU 策略假设高频访问数据在未来仍具有较高访问概率, 而 FIFO 则假设新加载数据更可能被再次访问. 在当前的搜索模式下, 查询过程中频繁访问的是新近加载的数据, 导致早期加载的数据逐渐被淘汰, 两种策略在效果上趋于一致. 然而, 当优先队列长度继续增加至 700–2 000 时, LFU 策略的命中率与性能优势逐步显现. 原因在于, FIFO 仅依据加载时间做出替换决策, 容易淘汰部分虽加载较早但仍具有较高访问频率的“历史热点”顶点, 进而降低了缓存利用效率. 而 LFU 能够更准确地保留频繁访问的关键顶点, 有效提升整体缓存命中率和搜索性能.

6.5 不同 k 值的影响分析

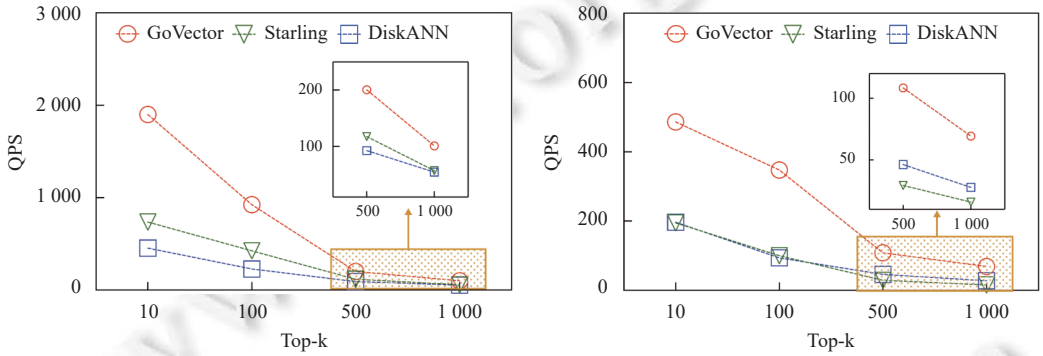
本节统计了 GoVector (即 GoVector-hybrid)、Starling 和 DiskANN 在不同 Top-k (k 取值为 10、100、500 和 1 000) 设置下, 在保证 99% 召回率前提下所达到的查询吞吐率 (QPS), 实验结果如图 12 所示. 实验结果如下.

当 k=10 时, GoVector 的 QPS 是 DiskANN 的 2.47–4.18 倍, 是 Starling 的 2.49–2.58 倍.

当 k=100 时, GoVector 的 QPS 是 DiskANN 的 3.69–4.03 倍, 是 Starling 的 2.18–3.46 倍.

当 k=500 时, GoVector 的 QPS 是 DiskANN 的 2.17–2.34 倍, 是 Starling 的 1.71–3.69 倍.

当 k=1000 时, GoVector 的 QPS 是 DiskANN 的 1.86–2.50 倍, 是 Starling 的 1.77–4.36 倍.



(a) DEEP (b) GIST

图 12 召回率为 99% 时不同 k 值对应的 QPS 表现

上述结果充分说明, 无论在何种候选集大小设置下, GoVector 均展现出较高且稳定的查询性能, 在确保高召回率的同时显著提升了系统的吞吐能力, 充分验证了 GoVector 在多样化检索精度需求场景中的通用性和高效性.

7 总 结

本文针对基于磁盘的向量近似最近邻搜索 (ANNS) 问题, 提出了一种基于向量相似性的 I/O 高效向量缓存策略, GoVector. 该方法依据 ANNS 搜索过程的两个阶段设计了静态-动态相结合的混合缓存机制: 静态缓存部分预加载入口顶点及其若干跳邻居, 用于加速初始阶段的近邻区域定位; 动态缓存部分在搜索过程中自适应地缓存候选顶点及其在向量空间中相似的邻近顶点, 以提升相似向量搜索阶段的命中率. 此外, GoVector 还引入了基于向量相似性的索引图重排序策略与自适应数据读取机制, 以增强缓存数据的空间局部性, 进一步提升查询吞吐率与 I/O 命中率. 实验结果表明, GoVector 在多个公开数据集上均显著优于当前主流的磁盘索引方法 (如 DiskANN 与 Starling), 展现出良好的性能优势. 然而, 混合缓存机制中静态与动态缓存的容量比例依赖于人工经验设定, 难以在不同查询负载和数据分布下实现最优配置. 在未来工作中, 我们计划引入查询感知与系统监测驱动的自适应缓存调整策略, 以自动调节静态与动态缓存的比例, 进一步提升混合缓存机制的通用性与鲁棒性.

References

[1] Asai A, Min S, Zhong ZX, Chen DQ. Retrieval-based language models and applications. In: Proc. of the 61st Annual Meeting of the Association for Computational Linguistics, Vol. 6 (Tutorial Abstracts). Toronto: ACL, 2023. 41–46. [doi: 10.18653/v1/2023.acl-tutorials.6]

[2] Xu YM, Liang HY, Li J, Xu ST, Chen Q, Zhang QX, Li C, Yang ZY, Yang F, Yang YQ, Cheng P, Yang M. SPFresh: Incremental in-place update for billion-scale vector search. In: Proc. of the 29th Symp. on Operating Systems Principles. Koblenz: ACM, 2023. 545–561. [doi: 10.1145/3600006.3613166]

[3] Grbovic M, Cheng HB. Real-time personalization using embeddings for search ranking at airbnb. In: Proc. of the 24th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. London: ACM, 2018. 311–320. [doi: 10.1145/3219819.3219885]

[4] Huang JT, Sharma A, Sun SY, Xia L, Zhang D, Pronin P, Padmanabhan J, Ottaviano G, Yang LJ. Embedding-based retrieval in facebook search. In: Proc. of the 26th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. New York: ACM, 2020. 2553–2561. [doi: 10.1145/3394486.3403305]

[5] Okura S, Tagami Y, Ono S, Tajima A. Embedding-based news recommendation for millions of users. In: Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Halifax: ACM, 2017. 1933–1942. [doi: 10.1145/3097983.3098108]

[6] Covington P, Adams J, Sargin E. Deep neural networks for YouTube recommendations. In: Proc. of the 10th ACM Conf. on Recommender Systems. Boston: ACM, 2016. 191–198. [doi: 10.1145/2959100.2959190]

[7] Lyu Y, Li ZY, Niu SM, Xiong FY, Tang B, Wang WJ, Wu H, Liu HY, Xu T, Chen EH. CRUD-RAG: A comprehensive Chinese benchmark for retrieval-augmented generation of large language models. ACM Trans. on Information Systems, 2025, 43(2): 41. [doi: 10.1145/3701228]

[8] Gao JY, Long C. High-dimensional approximate nearest neighbor search: With reliable and efficient distance comparison operations. Proc. of the ACM on Management of Data, 2023, 1(2): 137. [doi: 10.1145/3589282]

[9] Malkov YA, Yashunin DA. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2020, 42(4): 824–836. [doi: 10.1109/TPAMI.2018.2889473]

[10] Fu C, Xiang C, Wang CX, Cai D. Fast approximate nearest neighbor search with the navigating spreading-out graph. Proc. of the VLDB Endowment, 2019, 12(5): 461–474. [doi: 10.14778/3303753.3303754]

[11] Subramanya SJ, Devvrit, Kadekodi R, Krishaswamy R, Simhadri HV. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 1233. [doi: 10.5555/3454287.3455520]

[12] Chen Q, Zhao B, Wang HD, Li MQ, Liu CJ, Li ZZ, Yang M, Wang JD. SPANN: Highly-efficient billion-scale approximate nearest neighbor search. In: Proc. of the 35th Int'l Conf. on Neural Information Processing Systems. Curran Associates Inc., 2021. 398. [doi: 10.5555/3540261.3540659]

[13] Sun YF, Wang W, Qin JB, Zhang Y, Lin XM. SRS: Solving *c*-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. Proc. of the VLDB Endowment, 2014, 8(1): 1–12. [doi: 10.14778/2735461.2735462]

[14] Bentley JL. Multidimensional binary search trees used for associative searching. Communications of the ACM, 1975, 18(9): 509–517. [doi: 10.1145/361002.361007]

[15] Arora A, Sinha S, Kumar P, Bhattacharya A. HD-index: Pushing the scalability-accuracy boundary for approximate kNN search in high-dimensional spaces. Proc. of the VLDB Endowment, 2018, 11(8): 906–919. [doi: 10.14778/3204028.3204034]

[16] Wang MZ, Xu WZ, Yi XM, Wu SL, Peng ZY, Ke XY, Gao YJ, Xu XL, Guo RT, Xie C. Starling: An I/O-efficient disk-resident graph index framework for high-dimensional vector similarity search on data segment. Proc. of the ACM on Management of Data, 2024, 2(1): 14. [doi: 10.1145/3639269]

[17] MacAvaney S, Mallia A, Tonellotto N. Efficient constant-space multi-vector retrieval. In: Proc. of the 47th European Conf. on Information Retrieval. Lucca: Springer, 2025. 237–245. [doi: 10.1007/978-3-031-88714-7_22]

[18] Wang JG, Yi XM, Guo RT, Jin H, Xu P, Li SJ, Wang XY, Guo XZ, Li CM, Xu XH, Yu K, Yuan YX, Zou YH, Long JQ, Cai YD, Li ZX, Zhang ZF, Mo YH, Gu J, Jiang RY, Wei Y, Xie C. Milvus: A purpose-built vector data management system. In: Proc. of the 2021 Int'l Conf. on Management of Data. New York: ACM, 2021. 2614–2627. [doi: 10.1145/3448016.3457550]

[19] Fu C, Wang CX, Cai D. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2022, 44(8): 4139–4150. [doi: 10.1109/TPAMI.2021.3067706]

[20] Ge TZ, He KM, Ke QF, Sun J. Optimized product quantization. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2014, 36(4): 744–755. [doi: 10.1109/TPAMI.2013.240]

[21] Yin XZ, Gao C, Zhao ZJ, Gupta R. PANNS: Enhancing graph-based approximate nearest neighbor search through recency-aware construction and parameterized search. In: Proc. of the 30th ACM SIGPLAN Annual Symp. on Principles and Practice of Parallel Programming. Las Vegas: ACM, 2025. 369–381. [doi: 10.1145/3710848.3710867]

[22] Ram P, Sinha K. Revisiting *kd*-tree for nearest neighbor search. In: Proc. of the 25th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. Anchorage: ACM, 2019. 1378–1388. [doi: 10.1145/3292500.3330875]

[23] Ge TZ, He KM, Ke QF, Sun J. Optimized product quantization for approximate nearest neighbor search. In: Proc. of the 2013 IEEE Conf. on Computer Vision and Pattern Recognition. Portland: IEEE, 2013. 2946–2953. [doi: 10.1109/CVPR.2013.379]

[24] Jégou H, Douze M, Schmid C. Product quantization for nearest neighbor search. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2011, 33(1): 117–128. [doi: 10.1109/TPAMI.2010.57]

[25] Wang ZH, Yu DH, Li Q, Shen SG, Yao S. SR-HGN: Semantic- and relation-aware heterogeneous graph neural network. Expert Systems with Applications, 2023, 224: 119982. [doi: 10.1016/j.eswa.2023.119982]

[26] Johnson J, Douze M, Jégou H. Billion-scale similarity search with GPUs. IEEE Trans. on Big Data, 2021, 7(3): 535–547. [doi: 10.1109/TBDATA.2019.2921572]

[27] Feng XK, Peng YG, Cui JT, Liu YF, Li H. Locality sensitive hashing index based on optimal linear order. Chinese Journal of Computers, 2020, 43(5): 930–947 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.00930]

[28] Dasgupta A, Kumar R, Sarlos T. Fast locality-sensitive hashing. In: Proc. of the 17th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Diego: ACM, 2011. 1073–1081. [doi: 10.1145/2020408.2020578]

[29] Hassantabar S, Wang ZY, Jha NK. SCANN: Synthesis of compact and accurate neural networks. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2022, 41(9): 3012–3025. [doi: 10.1109/TCAD.2021.3116470]

[30] Cheng A, Chu D, Li T, Chan J, Crooks N, Hellerstein JM, Stoica I, Yu XY. Take out the TraChe: Maximizing (Tra)nsactional Ca(che) hit rate. In: Proc. of the 17th USENIX Symp. on Operating Systems Design and Implementation. Boston: USENIX Association, 2023. 419–439.

[31] Ahmed M, Seraj R, Islam SMS. The k-means algorithm: A comprehensive survey and performance evaluation. Electronics, 2020, 9(8): 1295. [doi: 10.3390/electronics9081295]

[32] Lei XF, Xie KQ, Lin F, Xia ZY. An efficient clustering algorithm based on local optimality of K-Means. Ruan Jian Xue Bao/Journal of Software, 2008, 19(7): 1683–1692 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1683.htm [doi: 10.3724/SP.J.1001.2008.01683]

[33] Jégou H, Douze M, Schmid C. Searching with quantization: Approximate nearest neighbor search using short codes and distance estimators. Technical Report, RR-7020, INRIA, 2009.

[34] Simhadri HV, Williams G, Aumüller M, Douze M, Babenko A, Baranchuk D, Chen Q, Hosseini L, Krishnaswamny R, Srinivasa G, Subramanya SJ, Wang JD. Results of the NeuriPS'21 challenge on billion-scale approximate nearest neighbor search. In: Proc. of the NeurIPS 2021 Competitions and Demonstrations Track. 2022. 177–189.

附中文参考文献

[27] 冯小康, 彭延国, 崔江涛, 刘英帆, 李辉. 基于最优排序的局部敏感哈希索引. 计算机学报, 2020, 43(5): 930–947. [doi: 10.11897/SP.J.1016.2020.00930]

[32] 雷小锋, 谢昆青, 林帆, 夏征义. 一种基于 K-Means 局部最优性的高效聚类算法. 软件学报, 2008, 19(7): 1683–1692. http://www.jos.org.cn/1000-9825/19/1683.htm [doi: 10.3724/SP.J.1001.2008.01683]

作者简介

周依杰, 博士生, CCF 学生会员, 主要研究领域为向量数据库, 图存储系统.

林圣原, 硕士生, CCF 学生会员, 主要研究领域为向量数据库, 图存储系统.

巩树凤, 博士, 讲师, 博士生导师, CCF 专业会员, 主要研究领域为向量数据库, 图存储与图计算系统.

余松, 博士生, 主要研究领域为向量数据库, 图存储系统.

范书豪, 本科生, 主要研究领域为向量数据库.

张岩峰, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为数据库, 大数据处理, 分布式系统.

于戈, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为数据库, 分布式系统.