arXiv:2611.29948v1 [cs.DB] 23 Nov 2026

三思而后行：借助 TriVS 重新架构磁盘驻留向量搜索

Yizou Chen[1] Baotong Lu[2] Peiqi Yin[1] Zhisheng Hu[1] Qi Chen[2] Ming-Chang Yang[ll]香港中文大学[2]微软研究院{

chenyz,zshu23,乘积量化yin22,mcyang}@cse.cuhk.edu.hk {baotonglu,cheqi}@microsoft.com

摘要

磁盘驻留近似最近邻搜索（ANNS）使向量搜索能够扩展到 DRAM 之外，但即便采用高速固态硬盘和复杂的 I/O 执行机制，最先进的系统仍然受 I/O 限制。为了解这一瓶颈，我们进行了以 I/O 为中心的分析，将每个查询分解为接近阶段（用于定位查询附近的区域）和收敛阶段（用于确定最终的 top-k）。分析揭示了互补的 I/O 权衡：基于聚类的搜索通过内存路由避开接近阶段的 I/O，但在收敛阶段会读取粒度较粗的完整簇；而基于图的搜索始终采用细粒度的向量级 I/O，从而实现高效收敛，但代价是在接近阶段不可避免地需要 I/O。

为结合两种粒度各自的输入/输出优势，我们提出了 *TriVS*，这是一种全新范式，将磁盘驻留向量搜索重新架构为三部曲：粗粒度接近、逐步蒸馏、精确收敛。其关键创新有两点。首先，它将接近阶段和收敛阶段从单一搜索粒度中解耦出来，采用基于聚类的搜索以避免接近阶段 I/O，并利用向量级 I/O 实现高效收敛。其次，为衔接这两种粒度，它在原有两个阶段之间引入了一个全新的内存蒸馏阶段，在访问磁盘之前，使用紧凑量化向量逐步将路由簇缩减为向量级候选列表。此外，更粗粒度的聚类会缩小质心路由索引，使 TriVS 能够将现有内存重新分配给蒸馏，而无需增加内存预算。我们在涵盖不同向量类型和规模的五个数据集上进行了评估，结果表明，在 Recall@10=0.9 下，TriVS 的吞吐量平均达到 3.42×PipeANN 的水平，延迟降低 51.2%，磁盘 I/O 减少 63.7%，同时使用的内存和存储均少于 PipeANN。

1 引言

现代搜索和检索系统将图像、文本及其他对象表示为高维向量。由于精确最近邻搜索在大规模数据集上成本过高

对于高维集合，近似最近邻搜索（ANNS）以少量召回率损失为代价，大幅减少所需的工作量 [26]。随着数据集增长到数十亿个向量，磁盘驻留近似最近邻搜索已成为事实上的高性价比向量搜索标准：它将全精度向量及其大型索引（每个通常需要数百吉字节 [10]）卸载到固态硬盘，同时仅在 DRAM 中保留紧凑的搜索状态。根据索引和磁盘驻留数据的组织与访问方式，现有磁盘驻留ANNS系统通常分为两类：1）基于聚类的近似最近邻搜索，以粗粒度的簇粒度管理向量（其中 SPANN [9] 是最先进的基线）；以及 2）基于图的近似最近邻搜索，以向量粒度将向量组织成近邻图（代表性系统包括 DiskANN [28]和 PipeANN [20]）。

尽管基于簇和基于图的搜索范式都已得到广泛采用并经过大量优化，但我们仍然观察到，即使使用速度更快的固态硬盘，它们在本质上仍受输入/输出限制，尤其是在高并发场景下。具体而言，我们在高端 PCIe 5.0 NVMe 固态硬盘上的测量结果表明，随着查询并发度增加，SPANN、DiskANN 和 PipeANN 都不再具备可扩展性：它们都逐渐达到固态硬盘的输入/输出容量，导致在高并发下，显式 I/O 等待占总查询延迟的 87–93%。尽管 PipeANN 采用了优化的流水线执行，这一瓶颈依然存在。这些结果表明，I/O 量是可扩展性的首要瓶颈，并据此确立了一项关键设计原则：可扩展的磁盘驻留近似最近邻搜索系统必须做到：最小化每查询磁盘 ***I/O***。

为了确定单查询输入/输出量的影响因素，我们将每次查询分解为接近阶段和收敛阶段：前者定位可能包含最近邻的区域，后者则检查该区域中的候选项，以确定最终的 top-k。我们的分析和实验表明，细粒度和粗粒度的搜索粒度在两个阶段各自带来互补的输入/输出优势。具体而言，基于聚类的搜索通过在内存中进行质心路由，避免了接近阶段 I/O；但在

收敛阶段会读取粗粒度的完整簇，因此产生过多的 I/O 量。相比之下，基于图的搜索在两个阶段都执行细粒度的向量级 I/O：这种细粒度检索有效限制了收敛阶段的 I/O 量，但接近阶段不可避免地会产生 I/O 开销。

为了结合已识别出的两种粒度各自互补的输入/输出优势，我们提出***TriVS***这一新范式，将磁盘驻留向量搜索重新架构为三部曲：粗略接近、渐进蒸馏、精确收敛。具体而言，TriVS 采用簇粒度以避免接近阶段 I/O，并利用向量粒度实现高效收敛。同时，为弥合两种粒度之间的巨大差距，TriVS 引入内存蒸馏阶段，在访问磁盘之前，使用紧凑量化向量逐步将路由簇缩减为向量级候选列表。这一三阶段工作流开辟了超越现有固定粒度图方法或基于簇的方法的第三条路径：随着搜索推进、查询空间缩小，它会自适应地利用逐步收缩的粒度，从而建立一种新颖的搜索感知方法，从根本上最大限度地减少每查询磁盘 I/O。

要使这一设计切实可行，必须在减少输入/输出与内存计算开销之间取得平衡。为满足这一要求，我们开发了 TriSearch，这是一种随着候选集缩小而逐步提高量化精度的向量搜索算法。它将渐进式细化与提前终止和下界剪枝相结合，以限制内存计算和精确向量 I/O。此外，TriVS 通过重新分配而非扩展内存来满足需求。通过支持更粗粒度聚类，内存蒸馏减少了质心数量并缩小了路由索引，从而释放 DRAM，以存放蒸馏阶段所需的量化向量。

我们在涵盖多种向量类型和规模的数据集上，对 TriVS 进行了全面评估。在五个数据集上，当召回率@10=0.9 时，TriVS 的吞吐量平均达到 PipeANN 的 3.42× 倍，延迟降低 51.2%，磁盘 I/O 减少 63.7%，同时使用的内存和存储也少于 PipeANN。

本文有三项贡献：

- 我们提出了以 I/O 为中心的分析，指出磁盘 I/O 是现有磁盘驻留近似最近邻搜索的可扩展性瓶颈，并揭示了细粒度和粗粒度搜索如何在探索阶段与收敛阶段之间转移 I/O（§3）。
- 我们将 TriVS 设计为一个三部曲，通过内存蒸馏解耦搜索粒度，并开发了 TriSearch 算法及其高效实现，以实现这一设计（§4）。
- 我们在多种工作负载和规模下对 TriVS 进行了全面评估，结果表明，通过减少磁盘 I/O，TriVS 的吞吐量高于最先进的磁盘驻留ANNS系统，延迟也更低（§5）。

2 背景

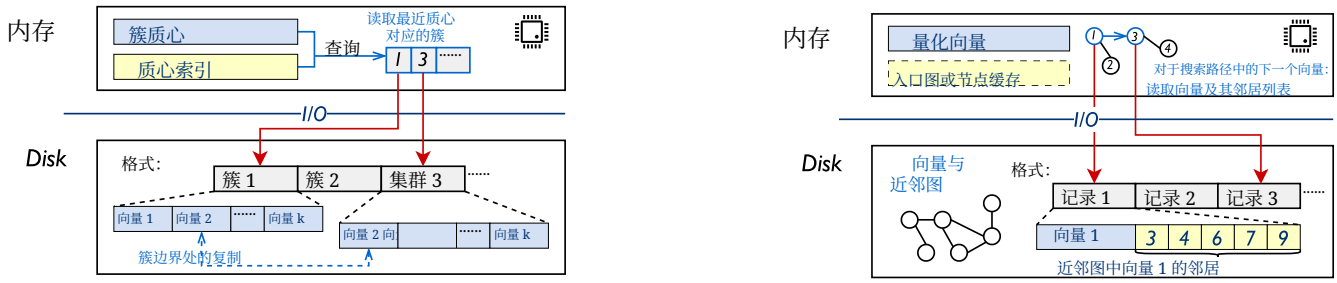
2.1 磁盘驻留近似最近邻搜索

给定一个向量数据集 X 和一个查询向量 q，向量搜索会根据选定的距离度量，检索 k 个 X 中与 q 最接近的向量。由于在高维数据集上进行精确搜索的成本高得难以承受，[26],现代向量搜索系统 [11,24,36] 依赖近似最近邻搜索（ANNS），以显著降低搜索成本，同时接受少量且可容忍的精度损失。ANNS 的精度通常以 Recall@k 衡量，即检索到的真实前k 个近邻所占的比例。随着数据集增长到数亿或数十亿个高维向量，仅全精度向量就可能占据数百 GB 的空间，而索引所需的容量也可能与之相当，甚至更大 [10,28,60]。因此，大规模向量搜索通常采用磁盘驻留近似最近邻搜索，将大部分向量和索引存储在固态硬盘上，同时将紧凑搜索状态保留在 DRAM 中 [9,20,25,28,48]。

现有磁盘驻留近似最近邻搜索系统可根据其组织索引和磁盘驻留数据的粒度进行分类。SPANN 等基于聚类的系统 [9] 采用粗粒度的簇粒度，而 DiskANN 等基于图的系统 [28] 和 PipeANN [20] 则采用细粒度的向量粒度。所选粒度会影响索引大小、数据放置、查询流程和磁盘访问，如图 1 所示。

基于聚类的磁盘驻留近似最近邻搜索。 SPANN 等基于聚类的系统 [9] 以簇为粒度组织向量：将彼此接近的向量分配到同一个簇，一并存储在磁盘上，并以该簇的质心作为表示（图 1(a)）。由于一个质心代表多个向量，路由索引包含的条目远少于逐向量图，因此可以常驻内存。查询时，系统先在内存索引中查找与查询向量最近的质心，再批量读取磁盘上对应的簇，并计算每个取回向量的精确距离。这种较粗粒度的组织方式消除了导航 I/O，并支持批量读取；但按簇粒度获取数据可能会传输所选簇中的所有向量，从而放大磁盘 I/O。为在限制簇探测次数的同时维持召回率，SPANN [9] 会在相邻簇之间复制边界向量，以增加存储开销为代价降低单查询输入输出。

基于图的磁盘驻留近似最近邻搜索。 DiskANN 等基于图的系统 [28] 和 PipeANN [20] 以向量粒度将向量组织成近邻图：每个顶点代表一个向量，边连接彼此接近的向量（图 1（b））。由于每个向量都连接到多个邻居，图的边整体规模可能与原始向量数据相当 [10,60]。在实际 DRAM 预算下，基于图的系统会将紧凑量化向量保存在内存中，并将每个精确向量与其图邻居列表一同存放在一个磁盘记录中，因此一次读取即可同时获取全精度向量和图边，以进行精确距离计算并继续遍历。



（a）基于聚类的近似最近邻搜索：按簇粒度组织　　**（b）基于图的近似最近邻搜索：**按向量粒度组织

图 1：**现有磁盘驻留近似最近邻搜索中的数据放置与查询流程**。基于聚类的系统通过内存中质心索引定位邻近簇，并将每个选定簇作为一个整体读取，从而避免导航 *I/O*，但可能会读取每个簇中的大量向量。基于图的系统在每条磁盘记录中存储一个全精度向量及其邻近图邻居，因此，扩展一个顶点（即向量）需要通过 *I/O* 读取其邻居列表，该列表用于引导后续搜索路径。

这些系统通常采用最佳优先搜索 [3, 49], , 反复扩展候选列表中距离最近且尚未访问的向量。每次扩展都会从磁盘读取精确向量及其邻居列表，计算该向量与查询之间的精确距离，并使用驻留在内存中的量化向量为其邻居打分并更新候选列表。与基于簇的方法相比，这种向量粒度的组织方式使单次读取更具选择性，并减少冗余数据访问，从而提高系统吞吐量；但由于图遍历中固有的依赖 I/O，查询延迟可能更高。当前最先进的 PipeANN [20] 会跨搜索步骤对读取进行流水线处理，并调整 I/O 并行度，使 I/O 与计算重叠，但仍保留相同的逐向量磁盘布局和访问模式。

2.2 向量量化

向量量化将高维向量映射为紧凑编码，以近似其与查询之间的距离。常见方法包括标量量化（SQ）、乘积量化（PQ）、 [32], OPQ、 [19], LVQ、 [1], 以及 RaBitQ [16, 18]。对于高维向量，SAQ [37] 会对 PCA投影向量进行分段，并为影响较大的维度分配更多比特；DADE [13], 、ADSampling [17],和 LeanVec [42] 则会减少或调整用于距离评估的维度。对于这些方法，不同的编码宽度、量化器配置和维度子集在准确性、内存占用和计算量之间实现了不同的权衡。DiskANN [28] 和 PipeANN [20]等磁盘驻留图系统会将 PQ编码保存在内存中，以引导向量级导航。从磁盘读取已扩展向量的邻居列表后，系统使用量化编码估计每个邻居到查询的距离，并更新有序候选列表，再据此选择下一个图记录。这种在内存中进行的距离估计无需从磁盘读取每个邻居的全精度向量。

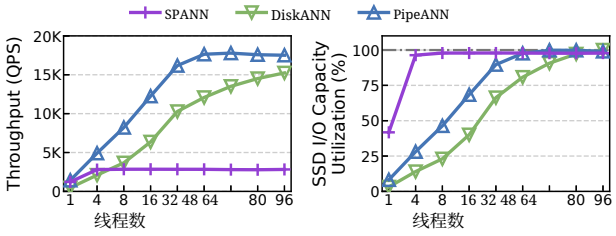
在这些方法中，RaBitQ [16, 18] 是一种先进且表现突出的量化器，可提供多个精度级别和理论误差界。给定一个参考中心 **c**，通常是一个聚类质心，RaBitQ 会对数据向量和查询的残差进行归一化，应用共享的随机旋转，并用一个或多个位表示每个残差维度。单比特编码提供初始估计，而额外的位平面会逐步细化估计，并复用先前的计算。在索引构建期间，RaBitQ 会存储每个向量的编码，以及根据该向量及其中心得出的若干预计算项。在精度级别 ℓ 下，搜索会将编码与变换后的查询残差进行计算，并结合这些项，得到估计距离 D_ℓ。RaBitQ 还会使用存储的误差项和查询残差范数推导出误差宽度 E_ℓ，以便在指定置信水平下，将精确距离界定在 $[D_\ell - E_\ell,\ D_\ell + E_\ell]$ 范围内。我们将相应的下界记为 $\mathrm{LB}_\ell = D_\ell - E_\ell$。

3 观察与动机：磁盘驻留近似最近邻搜索的 I/O 中心视角

本节从 I/O 中心视角探讨磁盘驻留近似最近邻搜索。我们首先指出，即使使用高端 PCIe 5.0 NVMe 固态硬盘，现有系统的扩展能力也会随着查询并发度的增加而迅速下降，并确定磁盘 I/O 是它们共同的瓶颈。为了解磁盘 I/O 的来源，我们将向量搜索过程分为搜索阶段和收敛阶段，并考察各阶段的操作粒度。分析表明，现有系统在整个向量搜索过程中采用统一的粒度，从而决定了各阶段磁盘 I/O 的数量和分布。

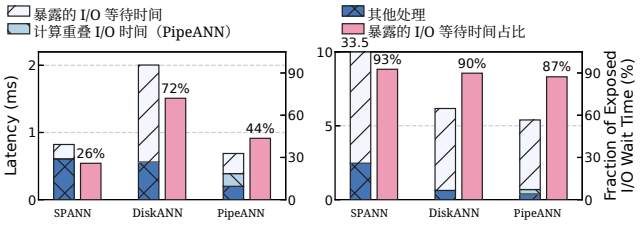
3.1 现有系统仍受 I/O 限制

我们首先测量 SPANN、DiskANN 和 PipeANN 在 DEEP-100 M 上的吞吐量可扩展性，Recall@10=0.9，使用同一块 PCIe 5.0 NVMe 固态硬盘上的 1–96 个查询线程。图 2(a) 显示，三个系统的扩展性均呈次线性：SPANN 仅用 4 个线程就达到吞吐量平台期，而 DiskANN 和 PipeANN 的吞吐量则逐渐低于初期扩展趋势。图 2(b) 将这种扩展行为与固态硬盘



(a) 查询吞吐量 (b) SSD I/O 容量利用率

图 2： **吞吐量扩展与 SSD I/O 利用率**。 在 *DEEP-100M* 上，当召回率为 *10=0.9* 时，随着各系统的 *SSD I/O* 容量接近饱和，其扩展能力均停止提升。



(a) 1 个线程 (b) 96 个线程

图 3： **DEEP-100M 上召回率为10=0.9 时的延迟分解与暴露的 I/O 等待时间**。 在 *1* 个线程下，*SPANN* 暴露的 *I/O* 等待时间较短，因为它无需经过遍历驻留磁盘的近邻图所需的依赖式 *I/O* 轮次；然而，在系统满负载（*96* 个线程）时，*I/O* 等待主导了所有三个系统的延迟。

输入/输出利用率联系起来。SPANN 仅用 4 个线程就达到固态硬盘输入/输出容量的 96% [1]，此时其吞吐量进入平台期。随着并发度增加，DiskANN 和 PipeANN 也逐渐接近容量上限，吞吐量增益随之递减。因此，即使使用高端 SSD，随着查询并发度推动固态硬盘输入/输出利用率接近容量上限，现有系统也无法维持吞吐量持续增长。

为考察这一 I/O 瓶颈如何影响每查询延迟，我们分解了延迟，并量化每次查询所暴露的磁盘等待时间，如图 3 所示。在 1 个线程下，SPANN 暴露的 I/O 等待时间占延迟的 26%，低于 DiskANN 和 PipeANN，这是因为它能够批量读取簇，并在图遍历中避免依赖式 I/O 轮次。尽管 PipeANN 优化了 I/O 执行，使部分 I/O 时间与计算重叠，但其暴露的 I/O 等待时间仍相当显著，占查询延迟的 44%。此外，在系统满负载（96 个线程）时，三个系统均由 I/O 主导，暴露的 I/O 等待时间占延迟的 87–93%。这些测量结果表明，I/O 量是可扩展性的首要限制因素，而依赖式 I/O 轮次则会在低负载下带来额外的延迟开销。

总而言之，无论是基于簇还是基于图的现有系统，在接近固态硬盘 I/O 容量时都会停止扩展。优化措施（例如计算与 I/O 重叠（如 PipeANN [20]））可以在低并发时隐藏部分 I/O 延迟，但不会减少单查询 I/O 量。因此，在高并发时，延迟和吞吐量仍受固态硬盘 I/O 容量限制。可扩展的磁盘驻留近似最近邻搜索必须遵循一项原则：最大限度减少每查询磁盘 ***I/O***。

3.2 不同的搜索粒度决定磁盘 I/O

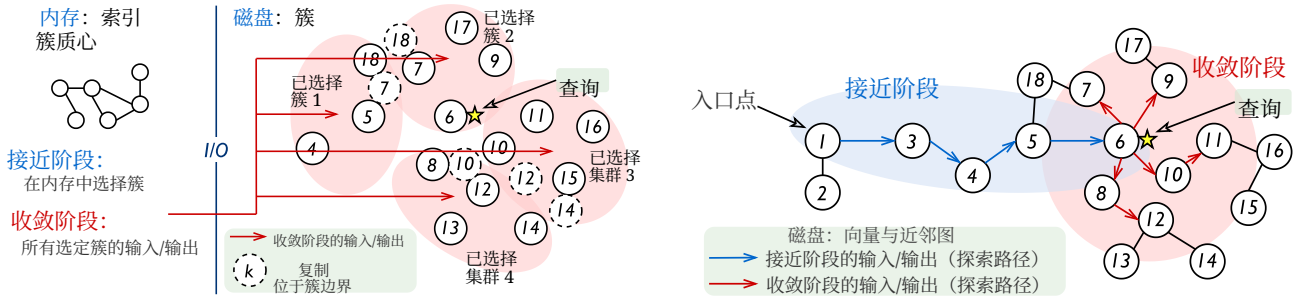
在确认磁盘 I/O 是共同瓶颈后，我们接下来考察它在向量搜索过程中是如何产生的。根据以往对向量索引遍历的分析 [20, 58, 60],，我们将每个查询分为两个分析阶段：接近阶段和收敛阶段。接近阶段将搜索从入口点引导至可能包含查询最近邻的区域。收敛阶段则检查该区域内的候选项，以确定最终的 top-k。我们将这一分析视角应用于基于簇和基于图的向量搜索，并研究各阶段产生的磁盘 I/O。

我们的分析和实验揭示了 I/O 过多的一个共同原因：现有架构将两个阶段都绑定到单一搜索粒度，优化一个阶段的同时牺牲了另一个阶段。这种不匹配源于两个阶段偏好的粒度不同：簇级路由使索引足够紧凑，从而可以驻留在内存中，消除接近阶段 I/O；而向量级访问则将收敛阶段 I/O 限制在选定向量上。

基于簇的磁盘驻留近似最近邻搜索消除了接近阶段的输入/输出，但代价是收敛阶段的输入/输出大幅增加。SPANN [9] 在两个阶段都采用簇粒度，如图 4(a) 所示。在接近阶段，其紧凑的质心索引完全在 DRAM 中选出查询近邻聚类，从而避免路由输入/输出。然而，在收敛阶段，系统会完整读取每个选中的簇，尽管其中只有一小部分向量能对最终结果有所贡献。图 5 量化了这一权衡：SPANN 不产生接近阶段输入/输出，但在收敛阶段每次查询传输 426 页，是 PipeANN 总输入/输出量的 7 倍。8×PipeANN 的总输入/输出量。

基于图的磁盘驻留近似最近邻搜索通过选择性读取向量来减少收敛阶段 I/O，但会产生存在依赖关系的接近阶段 I/O。 DiskANN [28]和 PipeANN [20] 在两个阶段都以向量为粒度运行，将每个向量视为近邻图中的一个顶点（图 4(b)）。在接近阶段，最佳优先搜索沿着向量级路径，从入口点朝查询目标前进；在收敛阶段，则继续扩展查询目标周围的单个节点。完整的向量级导航状态为每个向量包含一条图记录和一个邻接表，在实际服务预算下，其规模过大，无法全部保留在 DRAM 中，因此每次扩展都要读取一条包含全精度向量及其邻接表的磁盘记录。这种细粒度访问在收敛阶段具有选择性，因为搜索获取的是单个向量，而不是周围区域。在接近

[1]我们用固态硬盘 I/O 容量来表示每个系统在其访问粒度下可持续达到的最大读取 IOPS，该值使用 fio 测量 [2]。



（a）基于聚类的近似最近邻搜索：聚类级逼近与收敛阶段 **（b）基于图的近似最近邻搜索：**向量级逼近与收敛阶段图 4：**现有磁盘驻留近似最近邻搜索系统在整个搜索过程中采用单一粒度**。基于聚类的系统在内存中选择查询近邻聚类，并读取其中的每个向量；边界复制会进一步放大传输量。基于图的系统沿着指向查询的向量级路径前进，并扩展其周围的单个图记录。

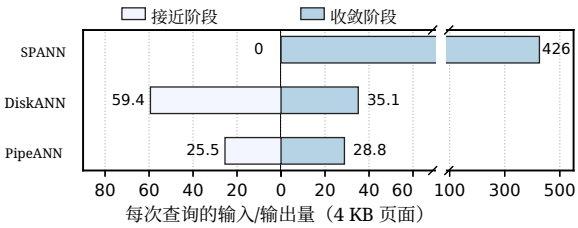


图 5：**现有架构会产生阶段特定的输入/输出**。*DEEP-100M* 在召回率*@10=0.9* 下。基于聚类的 *SPANN* 消除了接近阶段 *I/O*，但在收敛阶段传输的数据量大得多。基于图的 *DiskANN* 和 *PipeANN* 限制了收敛阶段 *I/O*，但会产生接近阶段 *I/O*。

阶段，然而，每次扩展都必须先读取当前向量驻留于磁盘的邻居列表，才能发现新的候选项，从而沿导航路径引入存在依赖关系的图记录 I/O。如图 5所示，DiskANN 和 PipeANN 在接近阶段每次查询分别读取 59.4 页和 25.5 页，随后在收敛阶段分别读取 35.1 页和 28.8 页 [2]。

总之，我们的分析表明，现有系统中的大量输入/输出源于将两个阶段耦合到单一搜索粒度。聚类级和向量级搜索粒度分别带来阶段特定的输入/输出成本，但也分别为接近阶段和收敛阶段提供互补的输入/输出优势。

4 TriVS

为降低接近阶段和收敛阶段的输入/输出成本，我们提出了 ***TriVS***。它通过将簇粒度分配给接近阶段、将向量粒度分配给收敛阶段，解耦两个阶段的搜索粒度；随后借助内存蒸馏阶段衔接这两个阶段，在进行精确向量输入/输出之前，将候选簇缩减为向量级候选列表。由此，TriVS 将驻留在磁盘上的向量

[2]对于每个查询，我们将最终 top-k（例如 top-10）中最近向量的精确评估完成作为阶段边界；在此之前完成的页面归入接近阶段，之后完成的页面归入收敛阶段。

搜索组织成一部三部曲：粗略接近、逐步蒸馏、精确收敛。§4.1推导粒度解耦及其内存实现机制；§4.2 将该设计映射到索引构建和数据布局；§4.3 介绍查询流程和 TriSearch 算法，该算法在整个三部曲中处理每个查询，同时限制内存计算和磁盘 I/O。§4.4 详述提升 TriVS 效率的优化，而 §4.5 讨论对在线更新的支持。

4.1 通过内存蒸馏解耦搜索粒度

解耦搜索粒度可以同时获得现有两种范式的输入/输出优势：聚类级逼近阶段避免了向量级图遍历输入/输出，而向量级收敛阶段保留了选择性读取，无需获取完整簇。然而，这两种粒度无法直接组合：聚类级逼近阶段识别候选簇，而向量级收敛阶段必须从其中选择少量向量。如果没有用于选择这一子集的内存机制，系统就必须完整读取并扫描选定簇，从而重新产生聚类级收敛及其粗粒度输入/输出。

我们观察到，紧凑量化向量可以保存在内存中，为聚类级逼近与向量级收敛之间提供了缺失的桥梁。即使在查询近邻聚类中，大多数向量仍然与查询足够远，可以利用基于这些内存表示计算出的近似距离估计进行剪枝。在完成簇路由后，TriVS 因此可以完全在内存中扫描量化向量，丢弃不太可能的候选项，仅保留向量级候选列表供收敛阶段使用。

这种内存中归约弥合了簇级方法阶段与向量级收敛阶段之间的粒度差距；我们将其称为蒸馏阶段。接近、蒸馏和收敛阶段共同定义了 TriVS，这是超越传统图式和基于簇设计的磁盘驻留近似最近邻搜索的第三种范式，

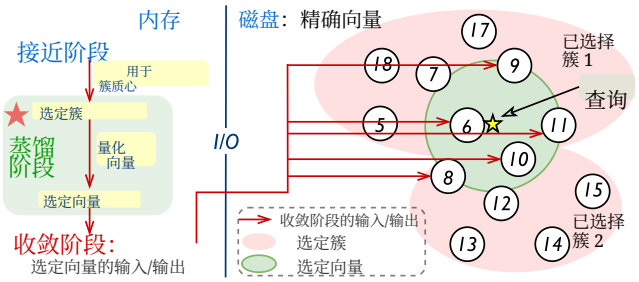


图6： **TriVS 通过内存蒸馏将搜索粒度解耦。** 接近阶段使用内存中质心索引选定簇；蒸馏阶段扫描这些簇中的量化向量，并生成向量级候选列表；收敛阶段只从磁盘读取候选列表中的精确向量，而不是读取选定簇中的所有向量。

以及基于簇的设计。如图 6所示，TriVS 将向量搜索组织成三部曲：首先以簇粒度接近查询附近的区域，然后完全在内存中将选定簇蒸馏为向量级候选列表，最后仅在磁盘上的候选短名单精确向量上进行收敛。与现有的基于图和基于簇的工作流（图 4（a）和 4（b））相比，TriVS 在接近阶段避免了向量级图遍历输入输出，也无需在收敛阶段读取完整簇。蒸馏阶段在这两种粒度之间提供了内存中桥梁，在进行精确向量 I/O 之前，将候选簇转换为单个候选项。

蒸馏重新分配内存，而非扩大内存。 自然会有人担心，使用内存中的量化向量进行蒸馏是否会增加内存使用量。事实并非如此：TriVS 重新分配现有内存以支持蒸馏，而不会扩大总内存占用。一方面，与基于图的系统相比，蒸馏不会增加新的逐向量证据：这些系统本就会将紧凑量化向量保留在内存中以指导搜索。PipeANN [20]将这些编码与采样入口图配对，而 TriVS 则将它们与大小相近的质心索引配对。另一方面，与基于簇的 SPANN 相比 [9],TriVS 可以扫描更多、更大的簇，因为评估其内存中的量化编码不会产生额外的完整簇磁盘读取。这种灵活性巧妙地支持了更粗粒度聚类，并消除了边界复制的需要。更粗粒度的聚类会产生更少的簇，因此质心也更少，从而缩小质心路由索引，为每个向量腾出一个紧凑编码的空间。将每个向量分配给一个簇，也进一步避免了这些编码被复制。通过这些重新分配，蒸馏得以改变搜索工作流程，同时不增加内存使用量。我们在 §5.2 中的评估证实，在 1亿和 10亿规模下，TriVS 的内存使用量都低于 PipeANN。

4.2 索引组织

借助内存蒸馏补上缺失的簇到向量接口后，TriVS 按阶段划分其服务状态：内存驻留结构支持路由阶段和蒸馏阶段，而全精度向量则保留在磁盘上，直到蒸馏确定候选短名单。图 7 展示了离线构建如何生成这种布局，以及查询在服务时如何使用它。

离线索引构建。 给定一个向量数据集，TriVS 首先采用大小受限的层次 k-均值聚类，目标簇大小为 T（在我们的实现中为 128 个向量），如图 7 顶部所示。具体而言，它会递归地对包含超过 $2k$ 个向量的任意簇应用均值聚类。TriVS 刻意采用粗得多的聚类方式：簇数量仅约为向量数量的 0.7–0.8%，而 SPANN 约为 10–12% [9]。这种更粗粒度的聚类既实用又有益：内存蒸馏无需完整簇磁盘读取即可扫描较大的簇，而较小的质心索引则为量化向量腾出更多内存。值得注意的是，在 TriVS 中，每个向量恰好属于一个簇，从而避免边界复制，降低内存和存储开销，同时简化构建。对于超出可用构建内存的数据集，TriVS 会一次处理一个子树。

聚类后，TriVS 会在最终的簇质心上构建内存路由索引（例如 HNSW [38]）。在每个簇内，它使用 RaBitQ [18] 相对于聚类质心对向量进行量化。按簇设置参考点可缩小残差分布范围，并提高蒸馏阶段的距离估计精度。最后，TriVS 将量化向量按簇连续存储在内存中，并按照相同的聚类簇顺序将全精度向量打包存储在磁盘上。

服务时的数据布局。在服务时，TriVS 会将质心索引、向量 ID、精确向量位置和 RaBitQ 编码保存在内存中，并按簇连续布局每个向量的状态。簇连续 RaBitQ 表示支持增量细化，同时每维仅占用两位。因此，查询和蒸馏阶段只访问 DRAM；只有在蒸馏完成候选短名单并解析每个候选的精确向量位置后，才会开始磁盘读取。

4.3 TriSearch 算法

我们设计了 TriSearch，这是一种向量搜索算法，可使每个查询依次经过 TriVS 的三阶段工作流：接近、蒸馏和收敛。在这些阶段中，TriSearch 同时限制内存中的计算量和精确向量 I/O。首先，簇级接近阶段搜索内存中质心索引，并返回最多 P 个查询近邻聚类。接下来，蒸馏阶段完全在内存中使用簇内连续存储的 RaBitQ 编码，逐步筛选选定簇中的向量，形成一个至多包含 R 个候选向量的候选短名单及其

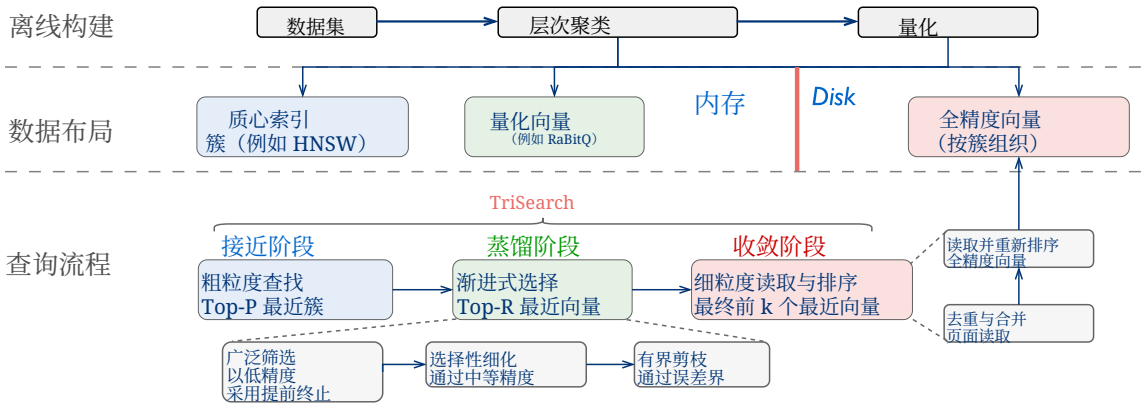


图 7：**TriVS 概览**。 离线构建会创建大小受限的簇，将每个向量分配到一个簇，并在内存中生成质心索引和按簇组织的量化向量，同时将按簇组织的全精度向量存储在磁盘上。搜索时，TriSearch 算法会路由到查询近邻簇，在内存中将其中的量化向量蒸馏为向量级候选列表，然后才读取入选的全精度向量并重新排序。

磁盘位置。最后，向量级收敛阶段仅从磁盘读取入选的全精度向量，并重新排序以返回最终的前 k 个结果。由于蒸馏会在磁盘 I/O 开始前确定所有精确向量访问，因此可以批量发出这些读取操作，避免依赖式 I/O 轮次，从而降低延迟。图 7 概述了这一三阶段工作流；算法 1 详细说明了完整的执行过程。

簇级方法阶段。 给定查询 q，TriSearch 首先使用标准 HNSW 搜索驻留于 DRAM 的质心索引 [38] 以定位查询近邻簇（算法 1，第 2 行）。搜索会返回一个有序集合C，其中至多包含 P 个簇；P 是簇探测预算，搜索宽度为 $ef = 1.2P$。这些路由簇将作为内存蒸馏的输入。此时，查询仅访问了质心；尚未读取任何驻留在磁盘上的向量。

渐进式内存蒸馏。 在接近阶段之后，路由簇仍包含大量向量。粒度解耦将从簇到向量的候选选择从磁盘读取转移到对 DRAM 中路由向量的量化距离评估。这一额外计算必须保持足够小，以免用 CPU 瓶颈取代 I/O 瓶颈。对每个路由向量使用更精确的量化表示进行评估代价过高，而仅使用粗略表示则会为精确向量 I/O 留下过多候选项。为平衡这些代价，TriSearch 随着候选集缩小而逐步提高量化精度：它首先使用 1位 RaBitQ 筛选每个路由向量，然后仅对有竞争力的候选项再读取一个位平面，以中等精度（2位 RaBitQ）对其进行评估。

为实现这一渐进过程，TriSearch 按顺序访问路由簇，并在其紧凑向量编码上调用 `ProgressiveScan` （第 5–6行）。对于每个向量块，1位RaBitQ评估会为每个向量计算低精度距离估计（第 17–18行）。TriSearch 维护一个按中精度距离（2位 RaBitQ）排序的 R 项最大堆 H。容量 R 限定了传递给精确访问选择的精化候选池大小。值得注意的是，RaBitQ 会将每个估计值与置信度控制下界配对，TriSearch 利用该下界来决定向量是否值得进行2位细化。在 H 填满之前，每个向量都有资格进行细化；之后，H中的最差中精度距离将成为准入阈值。只有当向量的1位 RaBitQ 下界低于该阈值时，TriSearch 才会计算其中精度距离；随后，如果精化后的距离使其进入当前前 R 项，便用该距离更新 H 项堆R（第20–22行）。

每一次成功的细化都会收紧后续向量的堆阈值，使后续筛选更具选择性。在各簇之间，堆未更新提供了提前终止信号：TriSearch 在W 个连续路由簇未能更新 H后停止，其中W 由 P （第5–7行）推导得出。簇内提升与簇级提前终止共同将日益精确的计算保留给日益缩小的候选集。

精确访问选择。 扫描终止时，H 中最多包含 R 个具有中精度距离和下界的候选项。TriSearch 使用近似前 S 个k 个候选项初始化精确访问候选列表S，并将其中第k小的距离作为其余候选项的阈值。随后，TriSearch 会保留中精度下界不超过该阈值的所有其余候选项，并添加存储在已为S选中的页面上的候选项，因为对这些候选项进行精确评估无需额外的页面读取（第 9–12 行）。

向量级收敛阶段（精确重排序）。候选短名单 S 确定后，TriSearch 会将其中的向量映射到磁盘页，对读取请求进行去重和合并，并随着所请求的页面到达，从全精度向量计算精确距离（第 13–14 行）。随后，它根据这些精确距离对候选短名单重新排序

算法 1 TriSearch 的伪代码。

输入： 对于查询 q，返回其前 k 个最近邻；
P：探测簇的最大数量；
R：重新排序的精确向量的最大数量（输入/输出）；
在精度级别 ℓ 下，D_ℓ 和 LB_ℓ 分别表示估计距离和距离的下界。

定义TriSearch(q, k; P, R)
// 1. 簇级方法阶段
  C ←路由簇(q,P)
  H ←容量为R的空最大堆，键为D_{mid}
  W ←从P导出的提前终止窗口
// 2. 内存蒸馏阶段
// 广泛筛选、选择性细化和排序
  对于所有 $c \in C$ 按路由顺序 **执行**
    更新 ←ProgressiveScan(q, c, H)
    如果 更新 = 对于W 个连续簇为 false**则**
      中断
// 精确访问选择
  $S \leftarrow k$ 个最佳候选项位于H中
  τ_k ←k个最小的D_{mid} 位于H中
  $S \leftarrow S \cup \{x \in H:$ 下界$_{\mathrm{mid}}(q,x) \le \tau_k\}$
  S ←添加相同页面候选项(S, H)
// 3. 向量级收敛阶段
  A ←页面规划(S)
  返回 ExactRerank(q,S,A,k)

defProgressiveScan(q, c, H)
  已更新 ←假
  对于所有 向量块 b，在簇 c **执行**
    $(D_{\mathrm{low}},\mathrm{LB}_{\mathrm{low}})$ ← 低级评估(q,b)
    对于所有 $x \in b$ **执行**
      如果 $\mathrm{LB}_{\mathrm{low}}(q,x)$ <阈值(H) **则**
        $(D_{\mathrm{mid}}[x],\mathrm{LB}_{\mathrm{mid}}[x])$ ← 中级评估(q,x)
        如果 UpdateHeap(H,x,$D_{\mathrm{mid}}[x]$)**则**
          已更新 ←true
  返回 已更新

并返回排名靠前的 k 个向量。

跨量化器的通用性。 TriSearch 并不局限于 RaBitQ。其低精度和中精度阶段可以采用不同量化方法中的不同编码宽度、量化器配置或维度子集。TriSearch 利用 RaBitQ 的理论下界来增强候选提升和剪枝，但不具备此类下界的量化器可以依赖估算距离、固定细化比例或特定于量化器的剪枝准则。不同的量化器可能影响准确性和效率，但仍保留 TriSearch 的三阶段工作流。

4.4 实现与优化

TriSearch 限制了搜索过程中各阶段的工作量；我们进一步优化了完整工作流程，以减少计算、输入/输出和内存占用。

高效的渐进式蒸馏。 对于蒸馏，每个簇都存储每维 1 位的 RaBitQ 基础编码，以及一个额外的位平面，从而构成嵌套的每维 2 位表示形式。这些编码按每块 32 个向量排列，以供 AVX-512 FastScan 内核使用 [14]。该内核会同时计算基础精度估计和下界；细化过程复用这些计算结果，只对额外的位平面进行求值。最后一个部分块仅存储有效通道，并在线程本地暂存空间中重建，以实现对齐访问，避免按簇填充。

紧凑的内存状态。 质心 HNSW 存储经 8 位标量量化（SQ8）的质心；Faiss [14] 支持这种配置，它适合用于识别一组广泛的候选簇。在每个簇内，TriVS 还会以 SQ8 编码每个向量的 RaBitQ 估计器字段。使用每个簇的质心作为 RaBitQ 的参考，可使其距离估计因子和界误差项的取值范围较窄，从而能够以 SQ8 编码，同时几乎不损失精度。对误差项进行保守舍入，可保留有效剪枝界。

页高效的向量级收敛。 全精度向量文件按照聚类簇顺序密集存储向量，使一次页面读取能够覆盖同一簇中的多个候选项。经过中精度门控后，TriVS 会纳入位于已选页面上的候选项，而无需额外读取；随后对页面 ID 去重，并将相邻页面合并为更大的读取。

内联和池化重排序。 精确重排序步骤可以在查询线程中内联执行，也可以交由专用线程池执行。在池化模式下，查询线程将最终确定的候选列表传递给专用重排序线程；随着先前请求完成，这些线程会持续发起新的精确读取。这种持续补充请求的方式可以使固态硬盘利用率更接近其每秒输入输出操作数上限。如果蒸馏后 I/O 仍是瓶颈，池化重排序会有所帮助，但也会带来交接和排队开销。因此，对于蒸馏阶段后仍受 I/O 限制的工作负载，TriVS 使用池化重排序；而当蒸馏阶段带来的 I/O 减少使压力转向计算时，则使用内联重排序。

4.5 讨论

TriVS 专注于搜索优化，而在线更新支持与其核心设计正交。以往工作将在线更新视为一个专门的设计问题：SPFresh [53] 支持基于聚类的磁盘驻留近似最近邻搜索系统中的更新；FreshDiskANN [41]、IP-DiskANN [52]，和 OdinANN [21] 支持基于图的磁盘驻留近似最近邻搜索系统中的更新。TriVS 可以采用 SPFresh 的局部再平衡机制：插入操作会将一个向量分配给其最近聚类，存储其簇内量化表示，并记录其精确向量的位置；而删除操作会使该向量的唯一条目失效。这种方法能够保留现有的查询流程，并简化更新处理，因为 TriVS 为每个向量仅分配一个簇且不进行复制，从而避免副本版本跟踪和清理。

5 评估

我们在多样化的工作负载和规模下，将 TriVS 与最先进的磁盘驻留近似最近邻搜索系统进行了评估。我们的主要发现包括：

• 在 Recall@10=0.9 下，TriVS 在五个数据集上的平均吞吐量达到 Pi-peANN 的 3.42× 倍，延迟降低 51.2%，磁盘 I/O 减少 63.7%。
• TriVS 的内存和存储占用与现有磁盘驻留近似最近邻搜索系统相当，且在大多数情况下更小。

• TriVS 逐步缩减查询工作；节省的 I/O 时间超过了蒸馏开销，并且其优势会随着查询并发度的增加而扩大。

5.1 实验设置

表 1： **评估中使用的数据集**。 维度。 表示向量维度，距离。 表示用于向量相似度的距离度量。

数据集	Dim.	类型 分布	向量数量	查询数量	大小（GiB）
DEEP-100M [54]	96	32位浮点数 L2距离	100M	10K	35.8
文本到图像 [5]	200	32位浮点数 内积	100M	100K	74.5
Wiki [50]	384	个32位浮点数 L2距离	100M	15.8K	143.1
SIFT-1B [33]	128	8位无符号整数 L2距离	1B	10K	119.2
DEEP-1B [54]	96	个32位浮点数 L2距离	1B	10K	357.6

数据集。 我们在五个公开数据集上对 TriVS 进行了全面评估，这些数据集涵盖多种向量类型和应用领域，规模从 1 亿到 10亿个向量不等，具有不同的维度和不同的距离度量（表 1）。具体而言，我们纳入了 BigANN基准测试中的热门数据集 [5],包括 DEEP-100M [54] 以及十亿规模的 SIFT-1B [33]和 DEEP-1B [54] 工作负载。我们还纳入了 Yandex文本到图像 [5],这一跨模态数据集，其基础向量源自图像，查询向量源自文本。基础向量和查询向量在共享嵌入空间中遵循不同的分布，因此构成一种分布外工作负载 [7]。最后，遵循 PipeRAG [31] 和现有工作 [55],，我们使用 BGE嵌入模型构建了一个名为 Wiki 的 384 维面向 RAG的数据集 [6], ，其中包含由维基百科文本数据生成的 1亿个基础向量 [50] 以及由 MMLU文本数据集生成的 15.8K 个查询向量 [22]。

基线系统。我们将*TriVS* 与三种最先进的磁盘驻留近似最近邻搜索系统进行比较：*DiskANN* [28],*Pi-peANN* [20],和 *SPANN* [9]。*DiskANN* 和*PipeANN* 使用 PQ 处理其驻留在内存中的量化向量。PQ 是它们的默认推荐配置，也是它们报告出色性能时采用的设置。由于*TriVS* 使用 RaBitQ，我们还纳入*PipeANN*官方的 RaBitQ 实现，记为*PipeANN+R*；该实现采用与 TriVS 相同的每维 2 位 RaBitQ 表示，以隔离量化方法的影响

。按照*PipeANN* [20],的设置，*DiskANN* 和*PipeANN* 索引使用$R = 96$ 作为图度，使用$L_{\text{build}} = 128$，并采用相匹配的 PQ 位宽（ DEEP 和 SIFT 为 32 字节，文本到图像为 50 字节，Wiki 为 96 字节）。查询时，*DiskANN* 使用束宽 4，以平衡吞吐量与延迟；而*PipeANN* 采用其默认策略，将 I/O流水线宽度动态增至最高 32。由于*PipeANN* 在内存中保留了一个占 1% 的采样入口图，因此其 DRAM 使用量高于*DiskANN*。为与*PipeANN*公平比较内存使用量，我们还纳入*DiskANN+C*，这是*DiskANN* 的一个变体，其节点缓存大小经过设置，约与*PipeANN*的内存使用量相当。

测试平台。所有实验均在一台服务器上运行，该服务器配备两颗 48 核 AMD EPYC 9455 处理器、384 GB DRAM，以及一块 Samsung 9100 PRO PCIe 5.0 NVMe 固态硬盘。使用 fio [2] 测量时，该固态硬盘在 4-KB 随机读取下的 IOPS 最高可达 100 万。操作系统为 Ubuntu 24.04.4，内核版本为 Linux 6.8.0。除非另有说明，实验均使用全部 96 个物理核心，并通过直接I/O访问固态硬盘。

TriVS 参数。我们将目标簇大小T 设为 128 个向量。TriVS 使用每维 2 位的 RaBitQ，使其每向量量化预算与 *PipeANN* [20],保持一致；后者的 PQ 表示约为每四个维度一个字节。我们将 RaBitQ 置信参数设为默认值 $\varepsilon_0 = 1.9$。我们联合调整簇探测预算P 和精确向量预算R ，并在每个召回率水平下保留性能最佳的配置，以获得 TriVS 在召回率与性能之间的帕累托前沿。不同的（P，R）组合可以通过在内存蒸馏和精确向量 I/O 之间转移工作量，实现相近的召回率。经验表明，在并发度较低时（例如 1 个线程），固态硬盘容量仍较充裕，TriVS 倾向于采用较小的P 和较大的R ；而在并发度较高时（例如 96 个线程），固态硬盘负载更重，TriVS 则倾向于采用较大的P 和较小的R 。对于每种配置，我们将提前终止窗口W 设为约.3P。默认情况下，TriVS 将总线程数的八分之一分配给池化重排序。对于计算密集型的 Wiki 数据集，以及线程数少于八个的配置，每个查询线程都会执行内联重排序。

性能指标。我们的主要性能指标包括查询吞吐量，以每秒查询数（QPS）衡量；查询延迟，以每次查询的平均处理时间衡量；以及磁盘 I/O 量，以每次查询读取的数据平均量衡量。我们还报告查询处理期间的内存使用量。遵循先前的工作 [9, 20, 28, 48],，我们使用 Recall@10 衡量搜索质量，并报告所有查询的平均 Recall@10。

5.2 总体性能

查询吞吐量和延迟。在召回率@10=0.9 时，图 8 的上、中两行显示，TriVS 在全部五个数据集上均实现了最高吞吐量和最低延迟。与 *PipeANN*相比，TriVS 的吞吐量提高了 3。42× 倍，平均延迟降低了 51.2%。

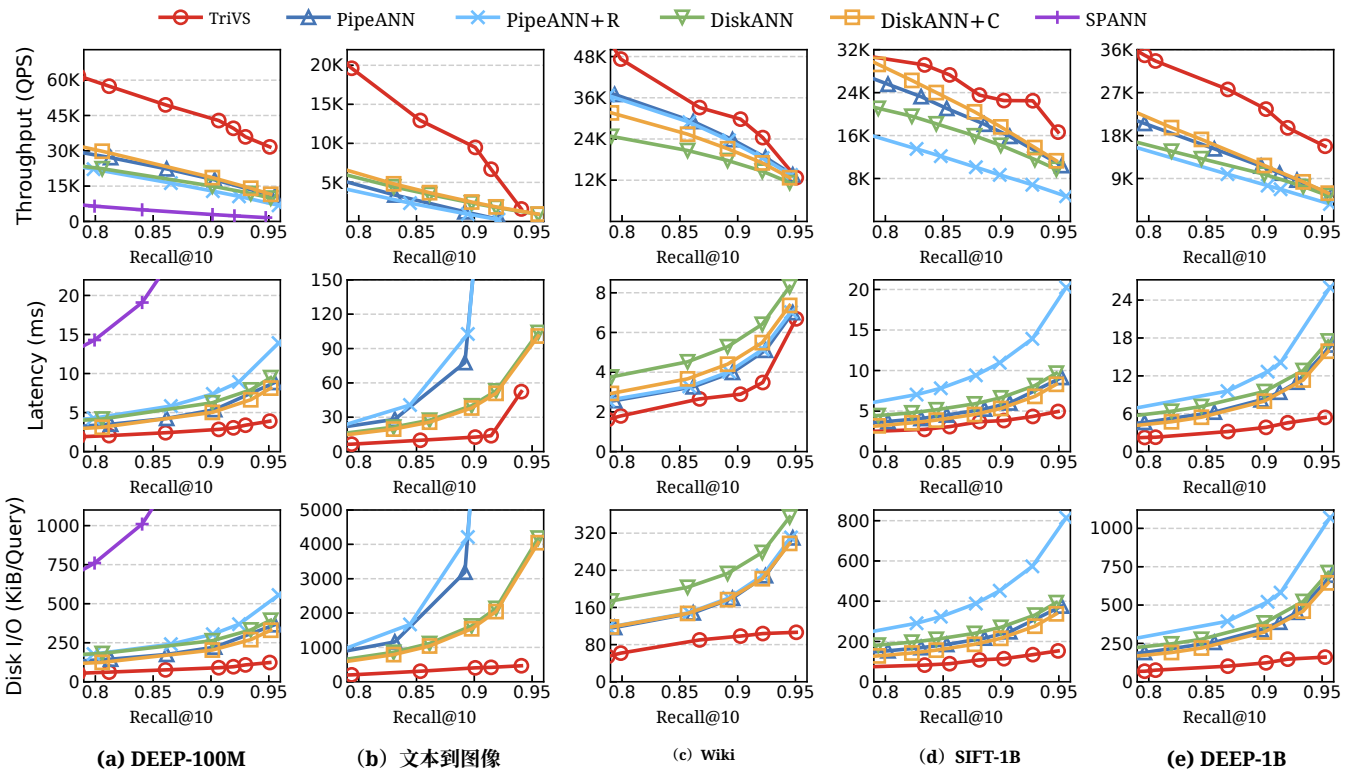


图 8： **总体性能。各列对应五个数据集；三行分别展示 Recall@10 变化时的吞吐量、延迟和每查询磁盘 I/O 量。***TriVS* 通过减少磁盘 *I/O* 提高吞吐量并降低延迟。

在每项工作负载的每个指标上，与最强基线相比，TriVS 的吞吐量提高了 1。29×–3。94× 倍（平均 2。18×倍），延迟则降低了 28.5%–68.1%（平均 44.8%）。这些优势在 1 亿规模和十亿级数据集上均成立，包括面向 RAG 的高维 Wiki 数据集，以及分布外的文本到图像工作负载。由于 *SPANN* 在我们规模最小的工作负载 DEEP-100M 上明显落后于其他基线，因此我们在更大规模或更高维度的工作负载中省略了它，这与以往的评估一致 [20, 48, 56]。

I/O 量是性能提升的关键。 图 8的底行显示，磁盘 I/O 越少，吞吐量和延迟表现越好。在 Recall@10=0.9 时，与每种工作负载下磁盘 I/O 量最低的基线相比，TriVS 将磁盘 I/O 减少了 47.3%– 74.7%（平均为 57.7%）；与 *PipeANN* 相比，平均减少了 63.7%。TriVS 在内存中路由并筛选候选项，然后仅读取候选短名单中的精确向量；基于图的方法在搜索过程中读取图记录，而 *SPANN*则在收敛阶段读取整个簇。节点缓存使 *DiskANN+C* 能够避免部分搜索阶段的读取，但无法彻底消除这些读取，因此仍存在与 *PipeANN*相同的结构性 I/O 瓶颈。RaBitQ 本身并不能解释这些性能提升：尽管它与 TriVS 使用相同的每维 2 位表示，*PipeANN+R* 的 I/O 量仍然更大，且在全部五种工作负载下都比 *PipeANN*慢。其粗粒度参考中心覆盖了不同的——

表2： **DEEP- 1亿/10亿上的查询服务内存和存储（GiB）**。由于构建所需资源超出可用内存，未列出 *SPANN* 在 *DEEP-1B* 上的数据。

	内存		存储	
系统	**100M**	**1B**	**100M**	**1B**
TriVS	4.14	42.0	42.98	430.0
PipeANN	5.91	43.4	80.20	798.2
PipeANN+R	6.27	59.3	81.41	814.1
DiskANN	3.02	29.9	79.31	792.8
DiskANN+C	6.09	43.5	79.31	792.8
SPANN	6.74	–	235.01	–

分散的图向量，而 TriVS 则对簇内的向量进行量化。

内存使用量。表 2 表明，这些性能提升并未以增加查询服务内存为代价。在 1亿和 10亿规模下，TriVS 的内存使用量分别比 *PipeANN*低 30% 和 3%，并且低于除 *DiskANN*之外的所有其他评估基线。TriVS 通过采用更粗粒度聚类实现了这一存储占用，这缩小了质心路由索引，并为紧凑的蒸馏编码释放了 DRAM。无需节点缓存时，*DiskANN* 仍是内存使用量最低的选择，能够在更紧张的 DRAM 预算下提供服务，但吞吐量会显著降低，延迟则会升高。

表3：**召回率@10=0.9 时每查询工作量的渐进式缩减**。低和中等精度下分别筛选和细化的低、中等精度计数向量；精确计数表示从磁盘读取、用于重排序的全精度向量数量。探测/扫描表示在搜索过程中选取的簇以及蒸馏过程中扫描的簇；提前终止会减少后者的数量，从而处理简单查询。

数据集	总计 #向量	探测/扫描 #簇	Low #向量	Mid #向量	精确 #向量
DEEP-100M	100M	688.0 / 459.8	66.0K	766.6	23.4
文本到图像	100M	3.07K / 2.82K	273.6K	13.4K	112.8
Wiki	100M	712.0 / 573.1	84.8K	641.2	25.4
SIFT-1B	1B	864.0 / 701.0	97.8K	1.03K	30.6
DEEP-1B	1B	768.0 / 621.1	89.4K	1.22K	32.4

存储使用量。此外，TriVS 的存储占用最小，比基于图的基线少约 46%，比*SPANN*少 82%。与基于图的系统相比，TriVS 无需存储逐向量邻接图。与 *SPANN*相比，TriVS 将每个向量恰好分配给一个簇，避免了边界复制造成的存储放大。TriVS 通过在粗粒度簇上进行内存蒸馏，保留了选择性精确读取的能力。

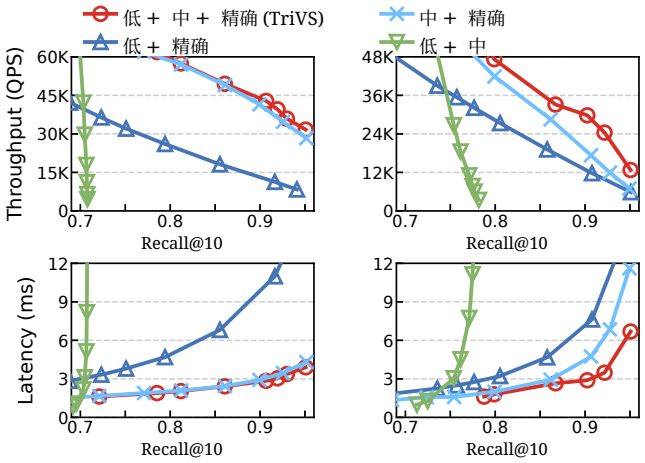
5.3 微观研究

接下来，我们将评估 TriVS 的关键设计选择和系统行为。除非另有说明，微观研究均使用 DEEP-100M，在召回率@10=0.9 下进行。

每个阶段都会逐步减少查询工作量。表3量化了各阶段在全部五个数据集上缩减查询工作量的程度。提前终止可避免扫描探测簇中 8–33% 的簇。中等精度仅评估低精度筛选出的向量中的 0.8–4.9%，而精确重排序只读取中等精度筛选后剩余向量中的 0.8–4.0%。总体而言，TriVS 在进行精确向量 I/O 之前，将候选集缩小了三个数量级以上。

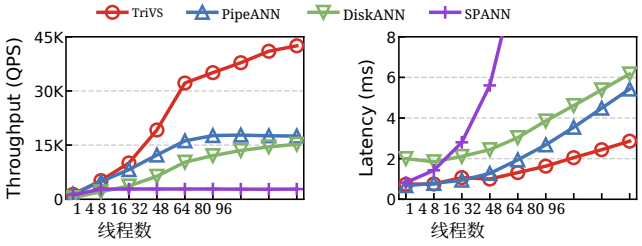
渐进式精度在计算、输入/输出和准确性之间取得平衡。表3中的减少量依赖于只在候选集缩小后才采用更高精度。图9比较了完整的 Low + Mid + Exact 流程与三种两级变体：Low + Exact、Mid + Exact 和 Low+ Mid。在 Recall@10=0.9 时，完整流程在 DEEP-100M 和 Wiki 上的表现分别比 Mid + Exact 高出 1.06× 和 1.58×，比 Low + Exact 高出 3.33×和 2.33× 。Low + Ex- act 会生成更大的精确短名单并产生更多输入/输出，因为其估计值较粗略；而 Mid + Exact 会以中等精度评估每个扫描向量，从而增加计算量，尤其是在 Wiki 上。

低精度 + 中等精度 无需精确向量 I/O，但若没有全精度重排序，就无法实现高召回率，在 DEEP-100M 和 Wiki 上的 Recall@10 分别停滞在 0.71 和 0.78。因此，三个层级缺一不可：低精度限制计算量，中等精度限制输入/输出，而精确重排序可实现高召回率。



(a) DEEP-100M (b) Wiki

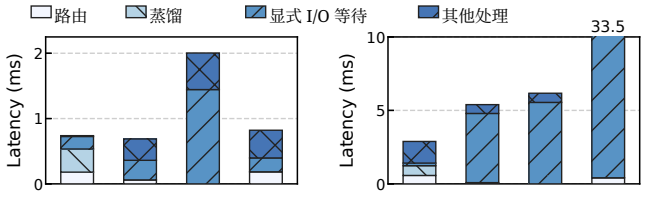
图9： **DEEP-100M 和 Wiki 上的渐进精度性能**。 低精度 + 精确搜索需要更大的精确短名单，因为其估计值较粗略；中等精度 +精确搜索需要更多计算，因为它会以中等精度评估每个扫描向量，在高维 *Wiki* 上尤其如此；低精度 + 中等精度无需精确 *I/O*，但无法实现高召回率。



（a）查询吞吐量 （b）查询延迟

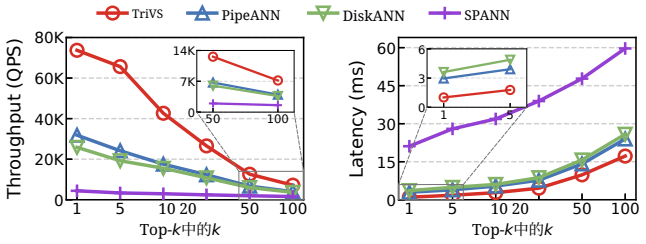
图10： **不同线程数下的吞吐量和延迟**。*TriVS* 在单线程时仍与 *PipeANN* 性能相近，并且在高并发下实现更高的吞吐量和更低的延迟。

查询并发度下的可扩展性。图10将 TriVS 纳入图2 中的基线扩展实验，并比较不同线程数下的吞吐量和延迟。单线程时，TriVS 的表现与 PipeANN 接近，吞吐量低 5%，延迟高 10%。随着并发度增加，基线逐渐接近图2所指出的 SSD 输入输出限制，TriVS 较低的单查询输入输出量使同一设备容量能够处理更多查询。TriVS 在约 32 个线程处扩展斜率的变化，部分反映了其对内存蒸馏和精确向量 I/O 的平衡方式：在召回率目标相同的情况下，SSD 容量充足、并发度较低时，TriVS 倾向于采用较小的P 和较大的R ；而在并发度较高、SSD 压力较大时，则倾向于采用较大的P 和较小的R 。在 96 个线程时，TriVS 的吞吐量达到 PipeANN 和 DiskANN 的 2.43× 倍和 2.79× 倍，同时将它们的延迟分别降低了 47% 和 54%。



(a) 1 个线程　　(b) 96 个线程

图 11： **1 个和 96 个线程下的延迟分解**。在高并发情况下，*TriVS* 的路由和内存蒸馏所耗时间被显著减少的输入/输出等待时间所抵消。



(a) 查询吞吐量　　(b) 查询延迟

图 12： **Top-k DEEP-100M 上召回率为 0.9 时的敏感性**。随着返回邻居数量增加，*TriVS* 仍能保持更高的吞吐量和更低的延迟。

节省的 I/O 时间超过蒸馏开销。图11 通过比较 TriVS 与 PipeANN 在单线程和 96 个线程时的延迟组成，解释了这一扩展趋势。单线程时，TriVS 在路由和蒸馏上耗时 0.53 毫秒，超过其节省的 0.11 毫秒显式 I/O 等待时间，因此延迟高出 7 %。然而，在 96 个线程时，TriVS 的路由和蒸馏耗时为 1.24 毫秒，却节省了 4.53 毫秒的显式 I/O 等待时间；这一减少抵消了其增加的内存内工作量，使端到端延迟降低了 46%。

Top-k 敏感性。 图 12 将 k（近似最近邻搜索返回的最近邻数量）从 1 调整到 100，并保持召回率@k=0.9。TriVS 在整个范围内实现了最高吞吐量和最低延迟，其吞吐量比 1.81×PipeANN 高，且在 k = 100时延迟低 28%。

池化重排序。 图 13 比较了在相同 96 核资源预算下的内联重排序与池化重排序。通过将查询执行与精确重排序解耦，池化机制会在读取完成时持续补充 I/O 队列，使 SSD IOPS 利用率从 69% 提升至 99%，吞吐量则从 31.5K QPS 提升至 42.1K QPS。

对集群大小的敏感性。 在图14中，我们改变目标簇大小 T 并保持目标召回率@10=0.9。默认值 T = 128 在查询吞吐量和内存使用量之间取得平衡。较小的簇会增加质心数量和路由开销，而较大的簇则需要进行更多内存扫描和精确向量读取才能保持召回率。同时，随着 T 增大，总内存用量趋近于实测的 3.83 GiB 非路由部分，因为紧凑的逐向量量化编码及其他服务状态不会随路由索引缩小。

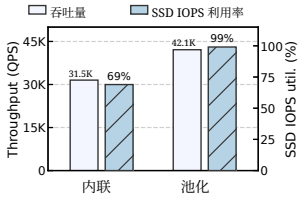


图 13： **内联与池化**

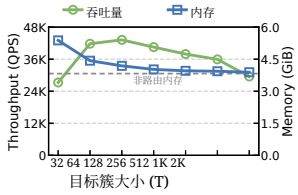


图 14： **对目标重排序的敏感性**。池化提高了**获取簇大小**。 T = 128 平衡 *SSD* 利用率、查询吞吐量和内存使用量。

6 相关工作

磁盘驻留近似最近邻搜索系统。 SPANN 等基于聚类的系统 [9] 使用内存中质心索引选择磁盘驻留的簇，但会完整读取并扫描每个选中的簇。Helmsman [25] 通过学习式集群剪枝（未公开发布）和专用 SSD 阵列执行来增强 SPANN，但仍采用单一簇粒度。

另一方面，基于图的系统保留了向量级选择性，但会在邻近图遍历期间从磁盘读取图记录，例如 DiskANN [28]。Starling [48]和 MARGO [57] 会重新排列图记录以改善页面局部性，但需要额外的构建时间和内存。PageANN [35] 构建页对齐图，使一次读取能够检查多个相邻向量。随着更高维度降低页面容量，这些局部性收益依赖于多个有用向量共享同一页面，并会逐渐减弱 [56]。面向缓存的设计包括 GoVector [62],，它将混合节点缓存与相似性感知重排序相结合；以及 GPS [56],，它面向高维向量，通过在内存和磁盘中优先存储邻接表而非完整向量来实现优化。PipeANN [20] 放宽严格的最佳优先搜索顺序，将相互依赖的图读取与计算进行流水线处理，使图搜索与 SSD 特性相匹配，从而实现先进性能。总体而言，页面感知布局、缓存和流水线执行能够减少图 I/O 或隐藏其延迟，但无法消除接近阶段 I/O。相比之下，TriVS 通过其内存中的接近阶段和蒸馏阶段消除了这种 I/O，仅在收敛阶段从磁盘读取入选的全精度向量。在页面感知放置和缓存与 TriVS 的收敛阶段 I/O 彼此正交，还可以进一步优化该 I/O；我们的评估表明，尽管 PipeANN 采用了流水线执行，TriVS 仍优于 PipeANN。

其他近似最近邻搜索系统。 传统的内存内近似最近邻搜索涵盖倒排文件 [4, 34], 基于图 [38], 基于树 [47],基于哈希 [12],和混合索引 [8]。除了仅使用 DRAM 的平台外，一些近似最近邻搜索系统还基于新兴的内存和存储技术构建，包括持久性内存 [40],计算快速链路连接的内存 [27], 内存处理 [51],和计算存储 [43]。基于加速器的系统利用图形处理器和现场可编程门阵列的专用并行能力来执行向量搜索 [23, 29, 30, 39, 44, 45, 59, 60]。分布式近似近邻搜索

系统将搜索扩展到多台机器上 [46,61],而解耦式系统则将计算与内存分离 [15,55]。TriVS 专注于磁盘驻留向量搜索，采用广泛可用的单节点 CPU、DRAM 和固态硬盘平台。

7 结论

我们提出了 TriVS，这是一种面向磁盘驻留近似最近邻搜索的新范式，通过内存蒸馏阶段，将方法阶段和收敛阶段与单一搜索粒度解耦。TriVS 将向量搜索组织成一个三部曲：先进行粗粒度搜索，逐步蒸馏，最后进行精确收敛；它结合了簇粒度和向量粒度互补的输入/输出优势，从而减少每查询磁盘 I/O。在各种工作负载下，TriVS 在不增加内存使用量的情况下，在吞吐量和延迟方面均优于最先进的系统。

参考文献

[1] Cecilia Aguerrebere、Ishwar Singh Bhati、Mark Hilde-brand、Mariano Tepper 和 Theodore Willke。借助压缩索引实现瞬时相似性搜索。《*VLDB*学报》，16(11)：3433–3446，2023年7月。

[2] Jens Axboe。fio：灵活的 I/O 测试器，2006年。

[3] Ilias Azizi、Karima Echihabi 和 Themis Palpanas。基于图的向量搜索：对最先进技术的实验评估。*ACM* 数据管理会议论文集，3(1)，2025年2月。

[4] Dmitry Baranchuk、Artem Babenko 和 Yury Malkov。重新审视十亿规模近似最近邻的倒排索引。载于欧洲计算机视觉会议，第 209–224 页。Springer，2018年。

[5] Big ANN Benchmarks 组织者。十亿规模近似最近邻搜索挑战：NeurIPS'21 竞赛赛道。https://big-ann-benchmarks.com/neurips21.html，2021年。

[6] Jianlv Chen、Shitao Xiao、Peitian Zhang、Kun Luo、Defu Lian 和 Zheng Liu。M3-embedding：通过自知识蒸馏实现多语言性、多功能性和多粒度文本嵌入，2025年。

[7] Meng Chen、Kai Zhang、Zhenying He、Yinan Jing 和 X. Sean Wang。Roargraph：用于高效跨模态近似最近邻搜索的投影二部图。《*VLDB*学报》，17(11)：2735–2749，2024年7月。

[8] Qi Chen、Haidong Wang、Mingqin Li、Gang Ren、Scar- lett Li、Jeffery Zhu、Jason Li、Chuanjie Liu、Lintao Zhang 和 Jingdong Wang。*SPTAG*：用于快速近似最近邻搜索的库，2018年。

[9] Qi Chen、Bing Zhao、Haidong Wang、Mingqin Li、Chuanjie Liu、Zengzhong Li、Mao Yang 和 Jingdong Wang。Spann：高效的十亿规模近似最近邻搜索。发表于第*35*届神经信息处理系统国际会议论文集，NIPS '21，美国纽约州雷德胡克，2021年。Curran Associates Inc.

[10] Rongxin Cheng、Yifan Peng、Xingda Wei、Hongrui Xie、Rong Chen、Sijie Shen 和 Haibo Chen。《揭示十亿级向量搜索中性能与索引大小之间的两难困境，并通过二级内存破解这一难题》，2024年。

[11] Paul Covington、Jay Adams 和 Emre Sargin。《用于 YouTube 推荐的深度神经网络》。载于第十届美国计算机协会推荐系统大会论文集，RecSys '16，第 191–198 页，美国纽约州纽约，2016 年。计算机协会。

[12] Mayur Datar、Nicole Immorlica、Piotr I ndyk 和 Va- hab S. Mirrokni。基于 p-稳定分布的局部敏感哈希方案。载于第二十届计算几何年度研讨会论文集，SCG '04，第 253–262 页，美国纽约州纽约，2004 年。计算机协会。[13] Liwei Deng、Penghao Chen、Ximu Zeng、Tianfu Wang、Yan Zhao 和 Kai Zheng。面向高维近似最近邻搜索的高效数据感知距离比较操作。《*VLDB*学报》，18(3):812–821，2024年11月。[14] Matthijs Douze、Alexandr Guzhva、Chengqi Deng、Jeff Johnson、Gergely Szilvasy、Pierre-Emmanuel Mazaré、Maria Lomeli、Lucas Hosseini 和 Hervé Jégou。Faiss 库。2024年。[15] Fei Fang、Yi Liu 和 Chen Qian。d-HNSW：基于解耦内存的高性能向量搜索引擎。*ACM* 测量、分析与计算系统论文集，10(2)，2026年5月。[16] Jianyang Gao、Yutong Gou、Yuexuan Xu、Yongyi Yang、Cheng Long 和 Raymond Chi-Wing Wong。面向近似最近邻搜索的欧几里得空间高维向量实用且渐近最优的量化。*ACM* 数据管理会议论文集，3(3)，2025年6月。[17] Jianyang Gao 和 Cheng Long。高维近似最近邻搜索：采用可靠高效的距离比较操作。*ACM* 数据管理会议论文集，1(2)，2023年6月。[18] Jianyang Gao 和 Cheng Long。RaBitQ：使用理论误差界量化高维向量

以用于近似最近邻搜索。*ACM* 数据管理会议论文集，2(3)，2024年5月。[19] Tiezheng Ge、Kai ming He、Qifa Ke 和 Jian Sun。优化乘积量化。*IEEE*模式分析与机器智能汇刊，36(4):744–755，2014年。[20] Hao Guo 和 Youyou Lu。通过使最佳优先搜索算法与 SSD 相匹配，实现低延迟的图式向量搜索。载于第*19*届 *USENIX* 操作系统设计与实现研讨会（*OSDI 25*），第 171–186 页，美国马萨诸塞州波士顿，2025年。USENIX协会。[21] Hao Guo 和 Youyou Lu。OdinANN：通过直接插入，在十亿级图式向量搜索中实现始终稳定的性能。载于第*24*届 *USENIX* 文件与存储技术会议（*FAST 26*），第 133–147 页，加利福尼亚州圣克拉拉，2026年2月。USENIX协会。[22] Dan Hendrycks、Collin Burns、Steven Basart、Andy Zou、Mantas Mazeika、Dawn Song 和 Jacob Steinhardt。大规模多任务语言理解的测量。国际学习表征会议论文集（*ICLR*），2021年。[23] Chengying Huan、Renjie Yao、Shaonan Ma、Rong Gu、Zhengyi Yang、Lizheng Chen、Zhibin Wang、Mingxing Zhang、Fang Xi、Guihai Chen 和 Chen Tian。Cmanns：通过计算与内存解耦实现 GPU 加速的近似最近邻搜索图索引构建。*ACM* 数据管理会议论文集，4(3)，2026年5月。[24] Jui-Ting Huang、Ashish Sharma、Shuying Sun、Li Xia、David Zhang、Philip Pronin、Janani Padmanabhan、Giuseppe Ottaviano 和 Linjun Yang。Facebook搜索中的基于嵌入的检索。载于第*26*届 *ACM SIGKDD*知识发现与数据挖掘国际会议论文集，KDD '20，第2553–2561 页，美国纽约州纽约，2020年。计算机协会。[25] Yuchen Huang、Baiteng Ma、Yiping Sun、Yang Shi、Xiao Chen、Xiaocheng Zhong、Zhiyong Wang、Yao Hu、Erci Xu 和 Chuliang Weng。聚类反击：借助 Helmsman（运维系统）大规模构建经济高效、高性能的近似最近邻搜索系统。载于第*20*届 *USENIX* 操作系统设计与实现研讨会（*OSDI 26*），第 1623–1639 页，美国华盛顿州西雅图，2026年7月。USENIX协会。[26] Piotr Indyk 和 Rajeev Motwani。近似最近邻：迈向消除维数灾难。载于第三十届 *ACM*计算理论研讨会论文集，STOC '98，第

第604–613页，纽约州纽约市，美国，1998年。计算机协会。[27] Junhyeok Jang、Hanjin Choi、Hanyeoreum Bae、Se- ungjun Lee、Miryeong Kwon 和 Myoungsoo Jung。CXL-ANNS：面向十亿规模近似最近邻搜索的软件-硬件协同内存解聚与计算。载于*2023 USENIX* 年度技术会议（*USENIX ATC 23*），第585–600页，波士顿，马萨诸塞州，2023年7月。USENIX协会。[28] Suhas Jayaram Subramanya、Fnu Devvrit、Harsha Vard- han Simhadri、Ravishankar Krishnawamy 和 Rohan Kadekodi。Diskann：单节点上的快速、精准十亿点最近邻搜索。载于 H. Wallach、H. Larochelle、A. Beygelzimer、F. d'Alché-Buc、E. Fox 和 R. Garnett（编），神经信息处理系统进展，第32卷。Curran Associates, Inc.，2019年。[29] Haodi Jiang、Hao Guo、Minhui Xie、Jiwu Shu 和 Youyou Lu。借助单个 GPU 实现高吞吐量、低成本的十亿规模向量搜索。*ACM* 数据管理会议论文集，3(6)，2025年12月。[30]Wenqi Jiang、Shigang Li、Yu Zhu、Johannes De Fine Licht、Zhenhao He、Runbin Shi、Cedric Renggli、S huai Zhang、Theodoros Rekatsinas、Torsten Hoefler 和 Gustavo Alonso。向量搜索的硬件与算法协同设计。载于高性能计算、网络、存储与分析国际会议论文集，SC '23，纽约州纽约市，美国，2023年。计算机协会。[31] Wenqi Jiang、Shuai Zhang、Boran Han、Jie Wang、Bernie Wang 和 Tim Kraska。PipeRAG：通过算法-系统协同设计实现快速检索增强生成，2024年。[32] Herve Jégou、Matthijs Douze 和 Cordelia Schmid。用于最近邻搜索的乘积量化。*IEEE* 模式分析与机器智能汇刊，33(1)：117–128，2011年。[33] Hervé Jégou、Romain Tavenard、Matthijs Douze 和 Laurent Amsaleg。搜索十亿个向量：使用源编码进行重排序。载于*2011*年 *IEEE* 声学、语音与信号处理国际会议（*ICASSP*），第861–864页，2011年。[34] Hervé Jégou、Romain Tavenard、Matthijs Douze 和 Laurent Amsaleg。搜索十亿个向量：使用源编码进行重排序。载于*2011*年 *IEEE* 声学、语音与信号处理国际会议（*ICASSP*），

pag第 861–864 页，2011年。

[35] Dingyi Kang、Dongming Jiang、Hanshen Yang、Hang Liu 和 Bingzhe Li。采用页对齐图的可扩展的基于磁盘的近似最近邻搜索，2025年。

[36] Patrick Lewis、Ethan Perez、Aleksandra Piktus、Fabio Petroni、Vladimir Karpukhin、Naman Goyal、Heinrich Küttler、Mike Lewis、Wen-tau Yih、Tim Rocktäschel、Se-bastian Riedel 和 Douwe Kiela。面向知识密集型自然语言处理任务的检索增强生成。载于 H. Larochelle、M. Ranzato、R. Hadsell、M. F. Balcan 和 H. Lin（编），神经信息处理系统进展，第 33 卷，第 9459–9474 页。Curran Associates, Inc.，2020年。

[37] Hui Li、Shiyuan Deng、Xiao Yan、Xiangyu Zhi 和 James Cheng。SAQ：通过代码调整和维度分段突破向量量化的极限。*ACM* 数据管理会议论文集，3(6)，2025年12月。[38] Yu A Malkov 和 Dmitry A Yashunin。使用分层可导航小世界图进行高效且稳健的近似最近邻搜索。*IEEE*模式分析与机器智能汇刊，42(4)：824–836，2018年。[39] Hiroyuki Ootomo、Akira Naruse、Corey Nolet、Ray Wang、Tamas Feher 和 Yong Wang。Cagra：面向图形处理器的高度并行图构建与近似最近邻搜索。载于 *2024*年第*40*届*IEEE*国际数据工程会议（*ICDE*），第 4236–4247 页，2024年。[40] Jie Ren、Minjia Zhang 和 Dong Li。HM-ANN：在异构内存上进行高效的十亿点最近邻搜索。载于 第*34*届神经信息处理系统国际会议论文集，NIPS 2020，美国纽约州雷德胡克，2020年。Curran Associates Inc.[41]Aditi Singh、Suhas Jayaram Subramanya、Ravis-hankar Krishnaswamy 和 Harsha Vardhan Simhadri。FreshDiskANN：用于流式相似性搜索的快速且准确的基于图的近似最近邻索引，2021年。[42] Mariano Tepper、Ishwar Singh Bhati、Cecilia Aguer-rebere、Mark Hildebrand 和 Ted Willke。LeanVec：通过使向量适配内存来加速向量搜索，2024年。

[43] Bing Tian、Haikun Liu、Zhuohui Duan、Xiaofei Liao、Hai Jin 和 Yu Zhang。使用 SmartSSD 实现可扩展的十亿级点近似最近邻搜索。载于*2024*年*USENIX*年度技术会议（*USENIX ATC 24*），第1135–1150页，加利福尼亚州圣克拉拉，2024年7月。USENIX协会。

[44] Bing Tian、Haikun Liu、Yuhang Tang、Shihai Xiao、Zhuohui Duan、Xiaofei Liao、Hai Jin、Xuecang Zhang、Junhua Zhu 和 Yu Zhang。通过 CPU/GPU 协同过滤与重排序实现高吞吐量、低延迟的十亿级向量搜索。载于第*23*届*USENIX*文件与存储技术会议（*FAST 25*），第171–185页，加利福尼亚州圣克拉拉，2025年2月。USENIX协会。

[45] Karthik Venkatasubba、Saim Khan、Somesh Singh、Har- sha Vardhan Simhadri 和 Jyothi Vedurada。BANG：使用单个 GPU 进行十亿规模的近似最近邻搜索。*IEEE*大数据汇刊，11(6):3142–3157，2025年。

[46] Jianguo Wang、Xiaomeng Yi、Rentong Guo、Hai Jin、Peng Xu、Shengjun Li、Xiangyu Wang、Xiangzhou Guo、Chengming Li、Xiaohai Xu、Kun Yu、Yuxing Yuan、Yinghao Zou、Jiquan Long、Yudong Cai、Zhenxiang Li、Zhifeng Zhang、Yihua Mo、Jun Gu、Ruiyi Jiang、Yi Wei 和 Charles Xie。Milvus：一款专为向量数据管理打造的系统。载于*2021*年国际数据管理会议论文集，SIGMOD '21，第2614–2627页，纽约，纽约州，美国，2021年。美国计算机协会。[47] Jingdong Wang、Naiyan Wang、You Jia、Jian Li、Gang Zeng、Hongbin Zha 和 Xian-Sheng Hua。用于近似最近邻搜索的三元投影树。*IEEE*模式分析与机器智能汇刊，36(2):388–403，2014年。[48] Mengzhao Wang、Weizhi Xu、Xiaomeng Yi、Songlin Wu、Zhangyang Peng、Xiangyu Ke、Yunjun Gao、Xiaoliang Xu、Rentong Guo 和 Charles Xie。Starling：一种高效输入/输出的磁盘驻留图索引框架，用于数据段上的高维向量相似性搜索。*ACM* 数据管理会议论文集，2(1)，2024年3月。[49] Mengzhao Wang、Xiaoliang Xu、Qiang Yue 和 Yuxiang Wang。基于图的近似最近邻搜索：全面综述与实验比较。《*VLDB*学报》，14(11):1964–1978，2021年7月。[50] 维基媒体基金会。维基媒体下载。https://dumps.wikimedia.org，2007年。[51] Puqing Wu、Minhui Xie、Enrui Zhao、Dafang Zhang、Ji ng Wang、Xiao Liang、Kai Ren 和 Yunpeng Chai。通过启用细粒度逐PIM核心调度，提升真实内存处理设备上的近似最近邻搜索性能。载于*2025*年 *USENIX* 年度技术会议（*USENIX ATC 25*），第1223–1241页，波士顿，马萨诸塞州，2025年7月。USENIX协会。

[52] Haike Xu、Magdalen Dobson Manohar、Philip A. Bernstein、Badrish Chandramouli、Richard Wen 和 Harsha Vardhan Simhadri。用于流式近似最近邻搜索的图索引原位更新，2025年。

[53] Yuming Xu、Hengyu Liang、Jin Li、Shuotao Xu、Qi Chen、Qianxi Zhang、Cheng Li、Ziyue Yang、Fan Yang、Yuqing Yang、Peng Cheng 和 Mao Yang。《SPFresh：面向十亿级向量搜索的增量原位更新》。载于第*29*届操作系统原理研讨会论文集，SOSP '23，第 545–561 页，纽约，纽约州，美国，2023 年。计算机协会。

[54] Artem Babenko（Yandex）和 Victor Lempitsky。《深度描述子十亿级数据集的高效索引》。载于*2016*年*IEEE*计算机视觉与模式识别会议（*CVPR*），第 2055–2063 页，2016 年。

[55] Peiqi Yin、Xiao Yan、Shiyuan Deng、Hui Li、Yifan Zhu、Xiangyu Zhi、Jingqi Mao、Ran Xu、Wenliang Zhang 和 James Cheng。《DistVS：采用计算-内存解耦的大规模向量搜索》。载于第*23*届*USENIX*网络系统设计与实现研讨会（*NSDI 26*），第 449–467 页，华盛顿州伦顿，2026年5月。USENIX协会。

[56] Peiqi Yin、Xiao Yan、Qihui Zhou、Hui Li、Xiaolu Li、Meiling Wang、Lin Zhang、Xin Yao 和 James Cheng。Gps：重新审视面向基于磁盘的高维向量搜索的数据布局。*ACM* 数据管理会议论文集，4(3)，2026年5月。

[57] Ziyang Yue、Bolong Zheng、Ling Xu、Kanru Xu、Shuhao Zhang、Yajuan Du、Yunjun Gao、Xiaofang Zhou 和 Christian S. Jensen。明智地选择边：面向基于磁盘的ANN搜索的单调路径感知图布局优化。《*VLDB*学报》，18(11):4337–4349，2025年7月。

[58] Qianxi Zhang、Shuotao Xu、Qi Chen、Guoxin Sui、Ji- adong Xie、Zhizhen Cai、Yaoqi Chen、Yinxuan He、Yuqing Yang、Fan Yang、Mao Yang 和 Lidong Zhou。VBASE：通过宽松单调性统一在线向量相似性搜索和关系查询。载于第*17*届*USENIX*操作系统设计与实现研讨会 *(OSDI 23)*，第377–395页，波士顿，马萨诸塞州，2023年7月。USENIX协会。

[59] Zili Zhang、Fangyue Liu、Gang Huang、Xuanzhe Liu 和 Xin Jin。《通过重新排序流水线，在超出 GPU 内存的大型数据集上实现快速向量查询处理》。载于第 *21* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 24*），第 23–40 页，加利福尼亚州圣克拉拉，2024年4月。USENIX协会。

[60] Haoru Zhao、Jingkai He、Jingyao Zeng、Mingkai Dong 和 Dong Du。《解耦图依赖关系，实现高效的十亿级 GPU 向量搜索》。载于第 *20* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 26*），第 1563–1584 页，华盛顿州西雅图，2026年7月。USENIX协会。

[61] Xiangyu Zhi、Meng Chen、Xiao Yan、Baotong Lu、Hui Li、Qianxi Zhang、Qi Chen 和 James Cheng。Cotra：借助远程直接内存访问实现高效且可扩展的分布式向量搜索。*ACM* 数据管理会议论文集，4(1)，2026年4月。

[62] Yijie Zhou、Shengyuan Lin、Shufeng Gong、Song Yu、Shuhao Fan、Yanfeng Zhang 和 Ge Yu。GoVector：一种面向高维向量最近邻搜索的 I/O 高效缓存策略，2025年。