DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node

摘要

目前最先进的近似最近邻搜索 (Approximate Nearest Neighbor Search, ANNS) 算法生成的索引通常需要存储在主内存中，以实现高查全率的快速搜索。然而，这种方法的高内存成本限制了数据集的规模。DiskANN 是微软提出的一种新的基于图索引的向量搜索系统，可以在单个只有 64GB RAM 和一个廉价的固态硬盘 (SSD) 工作站上，索引、存储和搜索十亿点数据库。由于将索引结构存储到磁盘上，所以搜索时不可避免的会发生多次随机磁盘 IO，DiskANN 通过减少搜索起点 (entry point) 到搜索终点的搜索跳步数，从而减少随机磁盘 IO 的次数。实验表明，DiskANN 建立的基于 SSD 的索引可以满足大规模 ANNS 的所有三个要求：高查全率、低查询延迟和高密度。然而，DiskANN 并未解决图索引本身存储局部性差的问题，这在一定程度上仍然制约了其性能提升。

本文在 DiskANN 的基础之上进行改进，提出一种新的数据存储结构模型 SpecANNS。相比于 DiskANN 使用的磁盘顺序存储，SpecANNS 采用了一种更具有存储局部性的存储结构，从而进一步减低搜索过程中随机磁盘 IO 的次数，显著优化了查询延迟，同时保持高召回率与高密度的点索引能力。实验表明，在 1M SIFT 数据集上，平均内存命中率比 DiskANN 提升 1.5 倍，平均搜索延迟减低了 20.3%；在 1M DEEP 数据集上，平均内存命中率比 DiskANN 提升 1.22 倍，平均搜索延迟减低了 22.8%。

关键词： 图索引的 ANNS；数据存储局部性；磁盘 IO 优化；大规模索引与检索

1 引言

向量检索是一项旨在寻找与给定查询向量最为接近的对象的过程。作为算法研究领域的核心议题，它在众多应用领域中扮演着至关重要的角色，涵盖了数据挖掘、信息检索、以及人工智能推荐系统等广泛领域 [13]。尤为值得一提的是，在近期大型语言模型 (large language models, LLMs) [4] 研究兴起的背景下，由向量数据库支持的神经网络服务已经成为现代人工智能基础设施的基本构建块。

如图 1 所示，例如文档、图像和演讲等等的不同格式的输入数据，均可以嵌入到高维向量空间中，并以特征向量的形式存储在向量数据库中。当聊天机器人接收到用户的查询请求时，将输入请求也嵌入高维向量空间中，并通过向量检索引擎迅速定位到与查询内容最为相似的一些对象。随后，将高度相关且富含上下文信息的结果将被传递给 LLMs 进行后续的深度处理与解读。

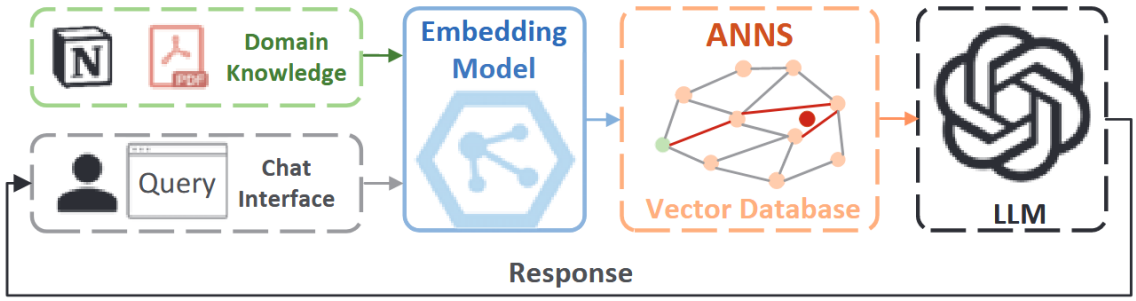


图 1. 向量检索在 LLMs 中的作用图

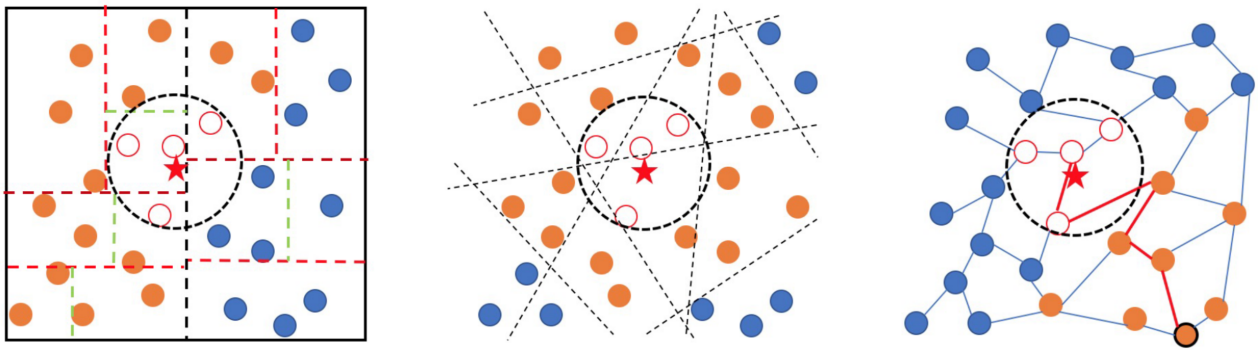
然而，为了在向量数据库中寻找与输入向量最接近的 K 个向量，一般只能通过遍历数据库中的所有向量，将输入向量与数据库中的每个向量都计算一次距离，并筛选出最短的 K 个距离所对应的那 K 个向量。由于维度诅咒，这种计算方式的时间复杂度过高，不能很好的适应现代大规模人工神经网络中的低延迟要求。为了满足大规模人工神经网络的高查全率、低查询延迟和高密度这三个要求，近似最近邻搜索 (Approximate Nearest Neighbor Search, ANNS) 通过牺牲一定的准确率，可以大幅提高查询的吞吐率以及降低搜索的延迟。

针对 ANNS 问题，研究者们已经开发出一系列多样化的算法，包括基于图结构 [11]、哈希索引 [7]、树形索引结构 [12] 以及量化 [9] 等方法。这些算法的核心差异体现在索引机制的设计上，每种方法都在吞吐量与准确率之间寻求独特的平衡点。大多数 ANNS 算法依赖于内存中的索引结构，以确保快速而精准的搜索性能，但这也导致了对内存资源需求的显著增加。例如，在像阿里巴巴这样的商业推荐系统 [6] 中，为了处理数十亿级别的向量数据，通常需要配置 TB 级的内存空间。然而，如此庞大的资源消耗大幅提高了总拥有成本 (total cost of ownership，TCO)，这不仅包括硬件采购费用，还涵盖了长期的运维支出。高昂的成本成为了限制 ANNS 服务扩展至包含数千亿向量的大规模数据集的主要障碍之一。此外，这些算法的可扩展性也受到了其索引结构复杂性的制约。维持庞大且精细的索引结构在主内存中，对于系统的资源管理提出了严峻挑战。随着数据规模的不断增长，如何有效降低内存占用，同时保持高效的搜索性能，已经成为该领域亟待解决的关键问题。面对这一挑战，探索更优化的索引策略和资源管理方案，是推动 ANNS 技术进一步发展的必然路径。

2 相关工作

ANNS 算法索引可以分为基于图索引和非基于图索引这两类方法，非基于图索引主要包括树结构方法、哈希技术方法和量化等方法。其中一些知名且广泛应用的算法，例如 KD 树 [15]、R* 树 [2]、VA-file [10]、局部敏感哈希 (Locality Sensitive Hashing，LSH) [5] 以及乘积量化 (Product Quantization，PQ)，都属于上述类别。部分研究专注于改进这些算法本身，而另一些则侧重于根据不同的平台和应用场景优化现有方法。

在最近的一些实验研究中，基于图的方法在性能上显著超越了某些知名的非基于图的方法 (如 KD 树、LSH、PQ)。这可能是由于非基于图的方法通过分割空间并为结果子空间建立索引来解决 ANNS 问题时，难以高效地索引这些子空间以快速扫描邻近区域，从而定位给定查询的最近邻。图 2 提供了一个例子，该图并未包含 PQ 算法，因为从某种角度来看，它可以被视为一种哈希方法。非基于图的方法为了达到高精度，需要检查许多邻近的单元格，导



(a) 树索引　　(b) 哈希索引　　(c) 图索引

图 2. ANNS 索引图

致大量远处的点被检查，这一问题随着维度的增加变得更加严重，即所谓的“维度诅咒”。相比之下，基于图的方法虽然可能从远离查询的位置开始，但通常能迅速接近查询，因为它们都是基于能够更好表达邻域关系的邻近图。

2.1 树索引

如图 2a所示，树索引结构所示树结构方法通过递归地将空间分割成子区域来组织数据点，其中红色五角星为查询的向量，黄色的点为访问过的点，以以五角星为圆心的虚线圆为查询到的范围，虚线圆内白色的节点为查询到的点，空间中的虚线为空间划分的结果。每次分割通常基于一个维度，并选择该维度上的中位数或其它标准值作为分割点，从而形成一系列节点和分支，直到所有数据点被分配到叶节点。查询时，算法遍历树，优先访问与查询点距离较近的节点。然而，随着维度增加，树结构的有效性急剧下降，因为每个节点需要检查的子节点数量呈指数增长，导致搜索效率低下。此外，对于动态数据集，维护树的平衡性和索引的准确性变得复杂且耗时，使得这种方法在处理大规模实时数据时不太适用。边界条件的处理也增加了计算负担，即当查询点接近两个或多个分区边界时，可能需要额外检查节点以确保结果的准确性。因此，在高维或大规模数据集中，树结构方法的局限性尤为明显。

2.2 哈希索引

如图 2b所示，哈希索引，特别是 LSH，旨在通过设计特殊的哈希函数使相似的数据点映射到相同的或相近的哈希桶中。这种映射不是基于数据点的实际位置，而是根据它们之间的相似度进行。为了提高搜索效率，通常使用多个哈希函数组合创建多层哈希表，减少误报率。尽管哈希技术能够在一定程度上缓解“维度诅咒”的影响，但它本质上是一个概率性的过程，无法保证找到确切的最近邻。由于是基于随机哈希函数的选择，可能会出现假阴性，即真正的最近邻未能被检索到。为了降低误报率，往往需要增加哈希表的数量或宽度，这直接导致内存占用上升。而且，哈希方法对参数调整非常敏感，不同的数据分布可能需要不同配置的哈希函数才能获得良好的性能。哈希方法难以适应动态数据集的变化，因为重新构建哈希表是一项昂贵的操作，尤其是在数据量庞大时，这些因素限制了哈希技术方法在实际应用中的灵活性和效率。

2.3 量化索引

量化索引，例如 PQ 和其他变种，通过将高维向量压缩为低维表示，以减少存储需求并加速搜索。具体来说，PQ 将向量空间划分为若干个子空间，然后在每个子空间内进行量化，用少量码字代表该子空间内的所有向量。查询时，计算查询向量与各子空间码字的距离，确定最相似的对象。量化方法的主要缺陷在于信息损失。由于高维向量被压缩成更紧凑的形式，原始数据的细节不可避免地丢失，这可能导致搜索结果精度下降。特别是在处理非均匀分布的数据时，某些区域的量化误差更大，进一步影响搜索质量。此外，量化方法依赖于预先定义的码书，构建高质量的码书是一个计算密集型任务，尤其是对于大规模数据集。而一旦数据分布发生变化，原有的码书可能不再适用，需要重新训练，这对系统的响应速度提出了挑战。虽然量化方法可以显著减少内存消耗，但它仍然需要额外的计算资源来进行编码和解码操作，这在一定程度上抵消了存储节省的优势。总的来说，量化方法在追求高效的同时，必须权衡精度损失和计算成本。

2.4 基于图索引的向量检索

另一种广受欢迎的 ANNS 搜索算法是基于图的方法。这类方法通过构建一个邻近图来实现高效的相似性搜索，其中每个候选向量被表示为图中的一个顶点，而两个顶点之间存在边则表明它们之间的距离足够近，如图 2c所示。在进行搜索时，算法从一个入口点开始，通过探索图中节点间的距离关系来查找最近邻。其核心思想是“邻居的邻居也可能是邻居”，即如果两个顶点与同一个顶点紧密相连，那么它们彼此之间也可能相当接近。

基于邻近图的方法能够在保证高精度的同时提供快速的搜索性能，这得益于图结构能够捕捉数据点之间的复杂关系，并利用这些关系加速查询过程。然而，这种方法并非没有缺点。首先，为了存储原始向量和额外的图结构，需要占用大量的内存空间，这对资源有限的系统构成了挑战。其次，图遍历过程中涉及大量的随机访问操作，这对于诸如 DRAM 和 SSD 等内存设备来说并不友好，因为这些设备在处理随机读写请求时效率较低，从而可能导致性能瓶颈。为了规避多次随机访问磁盘带来的高延迟问题，当前大多数基于图索引的 ANNS 算法均采用全内存处理方式。这种方法虽然显著提升了查询速度和准确率，但也伴随着对大量内存资源的需求，尤其是在构建和存储复杂的图索引结构时。单机系统往往难以承载大规模数据集所需的庞大内存容量，这使得精确的基于图索引的向量检索变得极具挑战性。然而，通过将图分割为多个子图，并采取分而治之的策略，可以有效地缓解这一难题。每个子图可以在分布式集群的不同节点上独立处理，然后将各子图的搜索结果进行合并，以获得最终答案。这种分布式处理方法不仅减轻了单个节点的存储压力，还保持了接近于全内存处理的检索速度。尽管如此，使用分布式集群意味着更高的硬件成本和运维开销，这对于许多应用来说是一个不容忽视的问题。

此外，邻近图的构建和维护本身也是一个复杂的任务。随着数据集的增长，更新或调整图结构的成本也会增加，尤其是在数据频繁变化的情况下。这不仅影响了系统的响应速度，还增加了运维成本。而且，尽管邻近图可以高效地捕捉局部邻域关系，但在处理全局结构或远距离相似性时，可能不如其他方法有效。因此，在选择是否使用基于图的 ANN 搜索算法时，必须权衡其在准确性、速度和资源消耗之间的利弊。

3 Vamana 图构造算法

DiskANN 通过将图索引存储到 SSD 中，使得只需要 64G 的 RAM 以及一根 SSD 即可完成大规模数据的向量检索。而由于 SSD 所带来的多次高延迟的随机磁盘 IO，DiskANN 通过尽可能的缩短起点 (entry point) 到搜索向量之间的搜索跳步数来减少随机 IO 的次数。

3.1 相对领域图与贪心搜索算法

大多数基于图的 ANNS 算法遵循以下工作方式：在索引构建阶段，它们根据数据集 P 的几何特性构建一个图 $G = (P\ E)$ 。到了查询阶段，对于给定的查询向量 x_q，搜索过程采用自然的贪婪策略或最佳优先遍历方法，在图 G 上进行。从某个指定的起点 $s \in P$ 开始，算法通过遍历图逐步接近查询向量 x_q。对于任何查询，人们已经做了大量工作来理解如何构建稀疏图，其中 $GreedySearch(s, x_q, k, L)$ 会快速收敛到近似的最近邻。至少当查询接近数据集点时，发生这种情况的充分条件是 NSG 中引入的所谓稀疏邻域图 (sparse neighborhood graph，SNG) [1]。在 SNG 中，每个点 p 的外邻域是这样确定的：初始化集合 $S = P \setminus \{p\}$。只要 $S \neq \emptyset$，就从 p 到 p* 添加一条有向边，其中 p* 是集合 S 中节点 p 的最近邻，并从 S 移除 $d(p, p') > d(p*, p')$ 的所有点 p'。由此不难看出，对于所有基点 $p \in P$ 和 $s \in P$，从任意 s 开始执行的贪婪搜索 $GreedySearch(s, x_p, 1, 1)$ 都可以收敛到 p。

虽然这种结构在原则上是理想的，但即使是中等大小的数据集也无法构建这样的图形，因为运行时间为 $\mathcal{O}(n^2)$，基于这种直觉，已经有一系列工作设计了更实用的算法，这些算法可以生成 SNGs 的良好近似值。但是，由于它们本质上都试图近似 SNG 属性，因此在控制这些算法输出的图形的直径和密度方面几乎没有灵活性。

3.2 Robust 剪枝算法

如前所述，满足 SNG 属性的图都是 GreedySearch 过程的良好候选者。然而，这些图的直径可能相当大，这会导致在磁盘上存储并查询这类图时出现大量的顺序读取操作。例如，在一维实数线上线性排列的点会形成一个 $\mathcal{O}(n)$ 直径的线图，其中每个点仅连接到它的两个邻居 (端点只有一个邻居)。这种结构虽然满足 SNG 属性，但在磁盘上进行搜索时，为了获取搜索路径中访问节点的邻居信息，将需要频繁地从磁盘读取数据，从而导致性能下降。

为了解决这个问题，我们希望确保在搜索路径上的每一个节点处，查询距离能够以乘法因子 $\alpha > 1$ 的方式减少，而不仅仅是按照 SNG 属性逐渐减小。考虑这样一个有向图，其中每个点 p 的出邻居由 `RobustPrune(p, V, α, R)` 过程决定。请注意，如果每个 $p \in P$ 的邻居都通过 $RobustPrune(p, P \setminus \{p\}, \alpha, n-1)$ 确定，那么从任意起点 s 开始的 `GreedySearch(s, p, 1, 1)` 将能够在对数级别的步数内收敛到目标点 $p \in P$，前提是 $\alpha > 1$。但是，这样做会使索引构建的时间复杂度达到 $\tilde{\mathcal{O}}(n^2)$，这是不理想的。

Vamana 在构建索引时采用了一种优化策略：它调用 `RobustPrune(p, V, α, R)`，但这里的 V 是经过精心挑选的，其包含的节点远少于 $n-1$。这一方法不仅减少了索引构建时间，还保证了在搜索过程中查询距离能有效地以 α 倍率递减，从而显著提高了搜索效率和性能。

具体来说，`RobustPrune` 过程旨在为每个点选择一组出邻居，使得从任何起点出发的搜索都能快速且有效地逼近目标点，同时尽量减少索引构建阶段的计算开销。通过这种方式，即使面对大规模数据集，DiskANN 也能实现高效的近似最近邻搜索，同时保持较低的磁盘 I/O

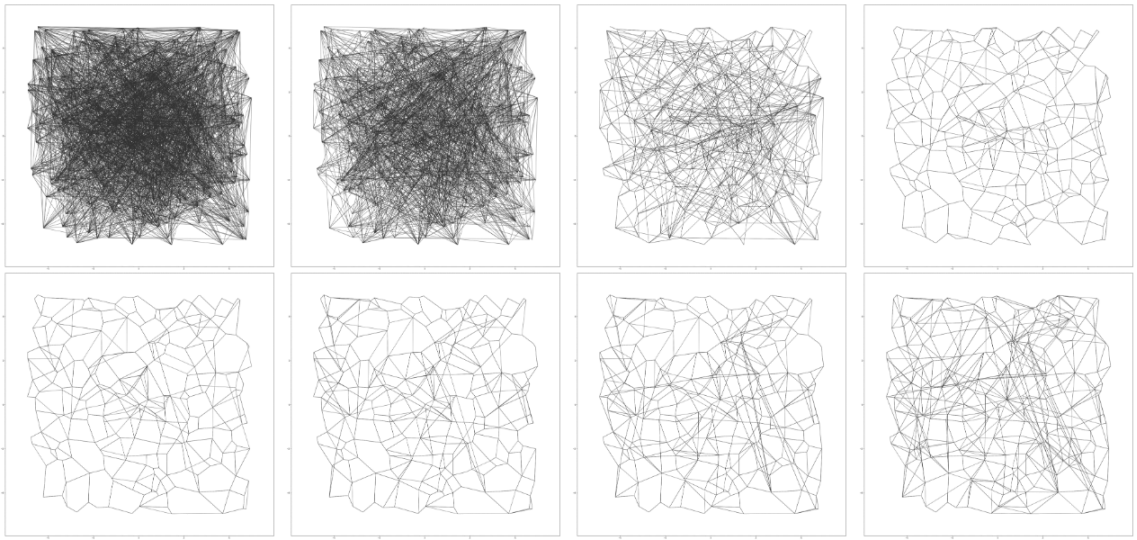


图 3. 图节点连接可视化图

成本。这种方法巧妙地平衡了搜索精度、速度与资源消耗之间的关系，为高维空间中的相似性搜索提供了强有力的工具。

3.3 Vamana 索引算法

Vamana 以迭代方式构建一个有向图 G。初始化时，每个顶点随机选择 R 个出邻居。需要注意的是，虽然当 $R > \log n$ 时图是良好连接的，但随机连接并不能保证 GreedySearch 算法收敛到好的结果。

接下来，令 s 表示数据集 P 的质心，它将作为搜索算法的起始节点。算法随后以随机顺序遍历所有点 $p \in P$，并在每一步中更新图，使得 $GreedySearch(s, x_p, 1, L)$ 更容易收敛到点 p。具体来说，在对应于点 p 的迭代中，Vamana 首先在当前图 G 上运行 $GreedySearch(s, x_p, 1, L)$，并将 V_p 设置为 GreedySearch 访问的所有点的集合。然后，算法通过运行 $RobustPrune(p, V_p, \alpha, R)$ 来确定 p 的新出邻居，从而更新图 G。

接下来，Vamana 通过为所有 $p' \in N_{\text{out}}(p)$ 添加反向边 (p', p) 来进一步更新图 G。这确保了搜索路径上访问的顶点与 p 之间存在连接，从而使更新后的图更适合 $GreedySearch(s, x_p, 1, L)$ 收敛到 p。然而，添加这种形式的反向边可能会导致 p' 的度数违反限制。因此，每当任何顶点 p' 的出度超过阈值 R 时，通过运行 $RobustPrune(p', N_{\text{out}}(p'), \alpha, R)$ 来调整图，其中 $N_{\text{out}}(p')$ 是 p' 的现有出邻居集合。

随着算法的进行，图逐渐变得更加适合和更快地支持 GreedySearch。我们的整体算法对数据集进行了两次遍历，可视化过程如图 3所示：第一次遍历时使用 $\alpha = 1$，可以快速的修剪图；第二次遍历时使用用户定义的 $\alpha \geq 1$。我们观察到，第二次遍历可以生成更优的图；但是，如果两次遍历都使用用户定义的 α，索引构建算法会变慢，因为第一次遍历计算出的图具有更高的平均度数，需要更多时间来处理，但第二次也剪枝过程中也会增加出很多适合快速搜索的长连接的边。

3.4 磁盘索引算法

总体思路非常直观：在数据集 P 上运行 Vamana 并将生成的图存储在 SSD 上。在搜索时，每当 GreedySearch 需要获取某个点 p 的出邻居信息时，我们只需从 SSD 中读取这些信息。然而，需要注意的是，仅存储十亿个点、每个点有 100 维的向量数据就会远远超过工作站的 RAM 容量！这提出了两个问题：如何在一个包含十亿个点的数据集上构建图，以及如果连向量数据都无法存储，如何在 GreedySearch 的搜索阶段进行查询点与候选列表中点之间的距离比较?

针对第一个问题的一种解决方案是使用聚类技术 (如 k-means) 将数据划分为多个较小的分片，为每个分片单独构建索引，并在搜索时仅将查询路由到少数几个分片。然而，这种方法会导致搜索延迟增加和吞吐量降低，因为查询需要被路由到多个分片。

与其在搜索时将查询路由到多个分片，为什么不将每个基点发送到多个附近的中心以获得重叠的簇呢? 具体来说，我们首先使用 k-means 将一个包含十亿个点的数据集划分为 k 个簇 (例如 $k = 40$)，然后将每个基点分配给最近的 ℓ 个中心 (通常 $\ell = 2$ 即可)。接着，我们为分配给每个簇的点构建 Vamana 索引 (此时每个簇大约只有 N/k 个点，因此可以在内存中完成索引构建)，最后通过简单合并边来将所有不同的图合并成一个图。实验证明，不同簇之间的重叠性提供了足够的连通性，使得 GreedySearch 算法即使在查询的最近邻分布在多个分片之间时也能成功收敛。值得注意的是，先前的工作 [3,14] 也探讨了通过合并多个较小的重叠索引来构建大规模数据集索引的方法，但它们构造重叠簇的方式不同，需要更详细的对比分析。

对于第二个问题，可以在主存中存储每个数据库点 $p \in P$ 的压缩向量 $\tilde{x}_p$，同时将图存储在 SSD 上。我们使用 PQ 对数据进行压缩，将数据点和查询点编码成短码，以便在 GreedySearch 的查询时间高效地获取近似距离 $d(\tilde{x}_p, x_q)$。需要指出的是，Vamana 在构建图索引时使用全精度坐标，因此能够有效地引导搜索过程朝向图中的正确区域，尽管在搜索时仅使用压缩数据。

将所有数据点的压缩向量存储在内存中，而将图以及全精度向量存储在 SSD 上。在磁盘上，对于每个点 i，我们首先存储其全精度向量 x_i ，然后是它的至多 R 个邻居的身份信息。如果一个节点的度数小于 R，我们会用零填充，以确保计算任何点 i 对应数据在磁盘中的偏移量是一个简单的计算过程，而无需在内存中存储这些偏移量。关于为何需要存储全精度坐标的解释将在下一部分中给出。

3.5 DiskANN Beam Search

一种自然的方法是运行 GreedySearch，根据需要从 SSD 中获取最近邻信息 $N_{\text{out}}(p^*)$。可以使用压缩向量来计算距离，以指导从磁盘读取最佳顶点 (及其邻域)。虽然这种方法合理，但需要频繁访问 SSD(每次访问需几百微秒)，从而导致较高的延迟。为了减少访问 SSD 的次数 (以顺序方式获取邻域) 而不过度增加计算量 (距离计算)，我们一次性获取 $L \setminus V$ 中最接近查询点 x_q 的少量 (例如 4 或 8 个) 点的邻域，并将 L 更新为这些候选点及其在此步骤中检索到的所有邻居中的前 L 名。需要注意的是，从 SSD 随机读取少量扇区的时间几乎与读取一个扇区相同。我们将这种修改后的搜索算法称为 BeamSearch。如果 $W = 1$，则该搜索类似于普通的贪婪搜索。注意，如果 W 太大 (例如 16 或更多)，则可能会浪费计算资源和 SSD 带宽。

尽管基于 NAND 闪存的 SSD 每秒可以处理 50 万次以上的随机读取，但要实现最大读

取吞吐量，必须使所有 I/O 请求队列饱和。然而，在高负载下操作会导致磁盘读取延迟超过一毫秒。因此，为了获得低搜索延迟，有必要在较低的负载因子下操作 SSD。我们发现，在较低的束宽 (例如 $W = 2, 4, 8$) 下操作可以很好地平衡延迟和吞吐量。在这种设置下，SSD 的负载因子在 30% 至 40% 之间，每个运行我们搜索算法的线程在 I/O 上花费大约 40% 至 50% 的查询处理时间。

4 复现细节

4.1 与已有开源代码对比

本文使用 DiskANN [8] 的开源代码进行改进，DiskANN 的顺序存储结构比较简单，但是局部性很差。本文在此基础之上，添加了扇区分配算法，并完成存储表的映射。

4.2 实验环境

我们使用系统为 Ubuntu 22.04.5 LTS 的服务器，其集成了带有 72M 缓存的 Intel Xeon Gold 5318 处理器，处理器主频为 2.10GHz，内存大小为 64G。

4.3 创新点

DiskANN 只是缩短了 entry point 到搜索向量的搜索跳步数，并没有改变图索引结构存储局部性差的缺点。在 DiskANN 中，节点是顺序存储的。不妨设每个扇区能存储 4 个节点，那么 0-3 号节点存储在第 0 号扇区，4-7 号节点存储在第 1 号扇区，以此类推。如图 4所示，节点相同颜色的即为存储在相同扇区。以第 0 号节点为例，其邻居为 7、14、8 中没有一个与节点 0 处于同一个扇区，邻居处于同一扇区中的比例为 0，这就说明了简单的顺序存储到磁盘中的局部性很差。但存储介质为磁盘时，由于顺序存储所带来的多次随机 IO 访问，将大大加剧了访问的延迟。我们使用节点 p 的邻居与 p 处于同一扇区的占比来衡量节点 p 的存储局部性，如下所示：

$$OR(p) = \begin{cases} \frac{|B(p) \cap N(p)|}{|B(p)-1|} & |B(p) > 1| \\ 0 & |B(p) \leq 1| \end{cases} \tag{1}$$

其中，B(p) 为节点 p 所在的扇区中的所有节点的集合，N(p) 为节点 p 的邻居集合。

我们提出一种新的存储结构，试图将节点 p 分配到其邻居所在最多的那个扇区，如果扇区存满了，那就顺延到邻居第二多的扇区，以此类推。当然如果节点 p 所有已分配的邻居的所在扇区都不可分配，那就随机把节点 p 分配到一个空的扇区即可。使用该算法对示例的图索引进行重新分配，其存储结构如图 5所示。以节点 0 为例，与节点 0 连接的 7、8、14 中，有 7 和 8 两个节点与 0 号节点处于同一扇区，其存储局部性肉眼可见的有所提升。

5 实验结果分析

我们在上文的环境中部署实验，并使用公开数据集 1M SIFT 与 1M DEEP 对 DiskANN 与 SpecANNS 进行测试。SIFT 数据集和 DEEP 数据集是计算机视觉领域中广泛使用的两个基准数据集，分别用于评估不同类型的特征提取和相似性搜索算法。1M SIFT 数据集包含 100

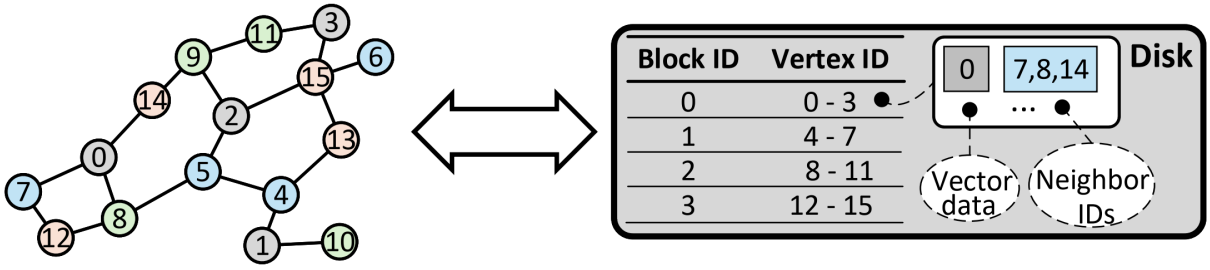


图 4. DiskANN 节点存储结构示意图

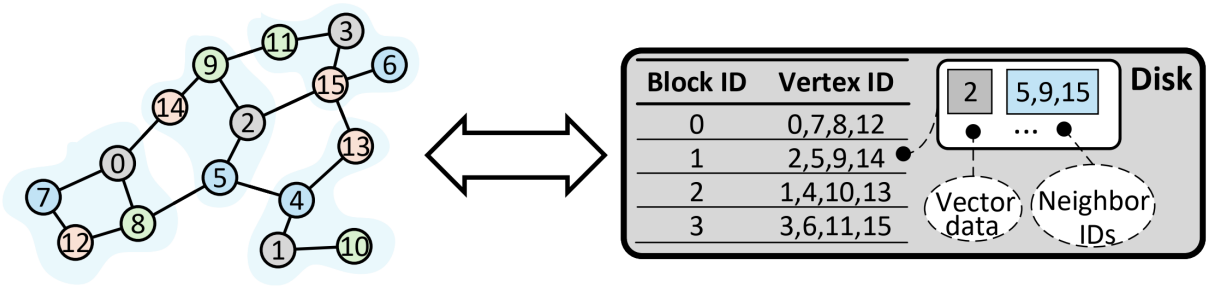


图 5. SpecANNS 节点存储结构示意图

万个 128 维的 SIFT 特征向量，这些向量是从大量自然图像中提取而来，经过尺度不变性和旋转不变性处理，能够捕捉图像中的局部区域特征。每个特征向量代表图像的一个局部区域，这使得它在进行相似性搜索时对算法提出了较高的要求，尤其是在高维空间中找到最近邻的问题上。

相比之下，1M DEEP 数据集由深度学习模型生成的特征向量组成，同样包含 100 万个特征向量，但这些向量是由预训练的卷积神经网络 (CNN) 从大规模图像分类任务的数据集中提取而来。通过移除 CNN 的最后一层分类器，得到的深层特征表示捕捉到了更高级别的语义信息，相比低级别的 SIFT 特征，它们更能反映图像的整体内容。尽管是深层特征，但由于使用了降维技术或特定的编码方法，特征向量的维度通常较低，有助于提高索引构建和搜索效率，并且在图像检索、人脸识别等视觉识别任务中表现出色。

1M SIFT 和 1M DEEP 数据集各自代表了不同类型特征向量的应用场景。前者强调局部特征的精确匹配，在图像特征检测方面表现突出；后者则侧重于全局语义的理解，在图像检索和分类任务中表现出色。这两个数据集都是 ANNS 实验评估中经常使用的公开数据集。

其中在 1M SIFT 数据集上，搜索过程中的内存命中率如图 6所示，由于 entry point 的 3 4 跳以内都在内存中，所以当搜索跳步数比较小的时候，两者的内存命中率都比较高。但是，SpecANNS 的内存命中率比 DiskANN 高。随着搜索跳步数的增大，beam search 的优势越来越小，因此二者的内存命中率都在下降。然而，SpecANNS 的命中率曲线一直在 DiskANN 的命中率曲线的上方，并且随着跳步数的增大，SpecANNS 比 DiskANN 的命中率比值越来越高。图 6显示，SpecANNS 的命中率最多比 DiskANN 的命中率多 1.5 倍。

在 1M SIFT 数据集上测试的召回率与平均搜索延迟如图 7所示，其中召回率为 recall@10，可以看到 SpecANNS 的曲线一直在 DiskANN 的上方，也就是如果对比同一召回率，SpecANNS 的搜索延迟比 DiskANN 低；如果是对比同一搜索延迟，SpecANNS 的召回率比 DiskANN 高。随着召回率的升高，SpecANNS 的搜索延迟比 DiskANN 的搜索延迟所下降的比例再增大，图 7中平均搜索延迟最多减少 20.3%。

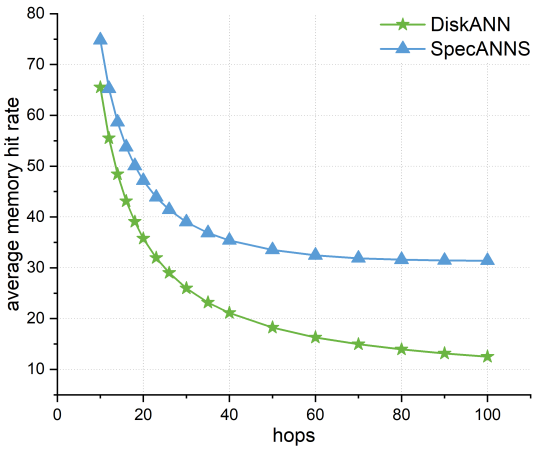


图 6. SIFT 数据集平均内存命中图

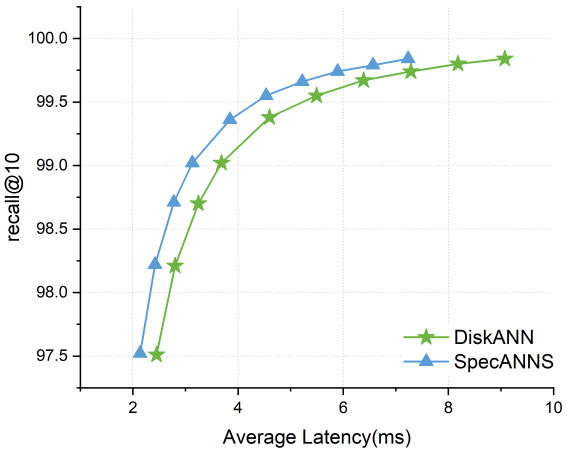


图 7. SIFT 数据集召回率与搜索延迟关系图

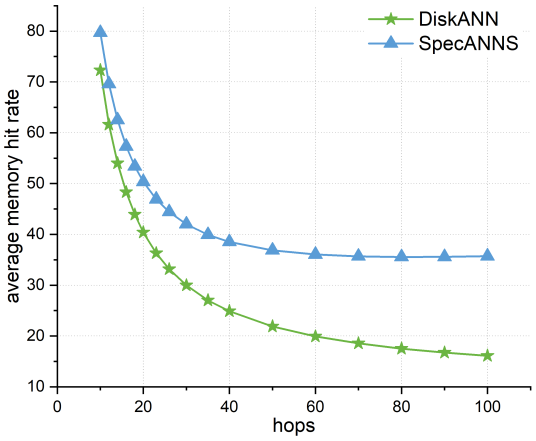


图 8. DEEP 数据集平均内存命中图

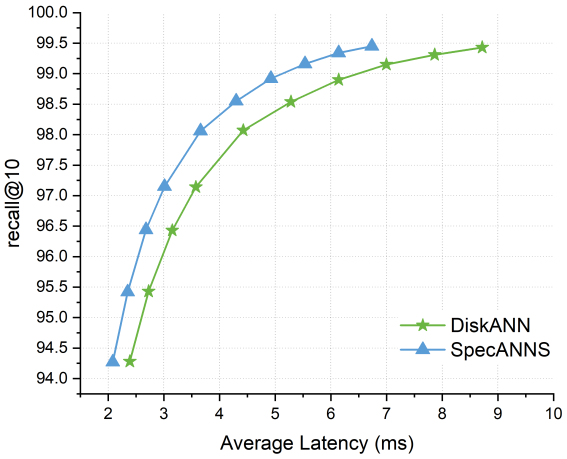


图 9. 数据集召回率与搜索延迟关系图

同理，我们在 1M DEEP 数据集上也进行了相同的测试，平均内存命中率如图 8所示，整体趋势与 SIFT 数据集解决，都是随着搜索步数的增加 SpecANNS 与 DiskANN 的平均内存命中率都在减小，而且 SpecANNS 的内存命中率比 DiskANN 的内存命中率提升的比例在增加，图 8中 SpecANNS 的平均内存命中率比 DiskANN 的平均内存命中率多 1.22 倍。

同理，我们也在 1M DEEP 数据集上测试了 recall@10 与搜索延迟的关系，如图 9所示，整体趋势与 SIFT 数据集一致，延迟随着搜索延迟的增加，召回率也在增加，但是 SpecANNS 的曲线一直在 DiskANN 的上方，即对比同一搜索延迟，SpecANNS 的 recall@10 比 DiskANN 的 recall@10 高；对比同一 recall@10，SpecANNS 的搜索延迟比 DiskANN 的搜索延迟低，图 9中，SpecANNS 的搜索延迟最多比 DiskANN 的搜索延迟减低了 22.8%。

6 总结与展望

在本文中，我们介绍了基于磁盘存储的图索引 ANNS 算法 DiskANN。为了将存储局部性差的图索引存储到磁盘，DiskANN 通过缩短 entry point 到终点之间的搜索跳步数，实现减少搜索过程中由于随机读写磁盘所带来的高延迟。但是 DiskANN 使用顺序存储结构，并没有结果存储局部性差的缺点。所以我们提出了 SpecANNS，这是在 DiskANN 基础之上添加了更具存储局部性的存储结构。实验表明，我们的 SpecANNS 比 DiskANN 的局部性要好，在 1M SIFT 数据集上，平均内存命中率比 DiskANN 提升 1.5 倍，平均搜索延迟减低了 20.3%；在 1M DEEP 数据集上，平均内存命中率比 DiskANN 提升 1.22 倍，平均搜索延迟减低了 22.8%。

基于磁盘存储的 ANNS 与全内存的 ANNS 相比，延迟还是很高，并不能做到量级上一致。所以我们可以使用存算一体架构来对其进行优化。例如可以在存算一体芯片中预测节点 p 的下一跳，然后先将该数据读取到内存中，那就减少由于随机磁盘 IO 所带来的高延迟的问题了。

参考文献

[1] Sunil Arya and David M Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280. Citeseer, 1993.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331, 1990.

[3] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[4] Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. Chatlaw: Open-source legal large language model with integrated external knowledge bases. *CoRR*, 2023.

[5] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[6] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*, 2017.

[7] Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. idec: indexable distance estimating codes for approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 13(9), 2020.

[8] 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.

[9] 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, 2010.

[10] T Michael Kashner. Agreement between administrative files and written medical records: a case of the department of veterans affairs. *Medical care*, 36(9):1324–1336, 1998.

[11] 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.

[12] Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[13] Bing Tian, Haikun Liu, Zhuohui Duan, Xiaofei Liao, Hai Jin, and Yu Zhang. Scalable billion-point approximate nearest neighbor search using {SmartSSDs}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 1135–1150, 2024.

[14] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. Scalable k-nn graph construction for visual descriptors. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1106–1113. IEEE, 2012.

[15] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. *ACM Transactions on Graphics (TOG)*, 27(5):1–11, 2008.