arXiv:2611.32908v1 [cs.OS] 24 Nov 2026

KuroANN：避免高召回率过滤向量搜索中的不必要 SSD 读取

Soobin Cho[1], Nakyung Lee[1],Jiwoong Park[2],Gyuyeong Kim[ll]延世大学 [2]诚信女子大学

摘要

我们介绍 KuroANN，这是一种面向分区索引的新型基于 SSD 的过滤向量搜索系统。现有解决方案为给定选择率范围内的所有查询分配相同数量的 SSD 读取次数，导致许多查询产生不必要的输入/输出开销。KuroANN 则根据通过查询过滤器的向量数量（称为匹配数量）为每个查询分配 SSD 读取次数。此方法基于我们的观察：匹配数量在搜索开始前已知，而且查询所需的 SSD 读取次数取决于该匹配数量。KuroANN 通过离线测量，确定在每种匹配数量下，为达到目标召回率需要探测的分区数量和读取的候选项数量。运行时，查询会查找这些数值，仅过滤已探测的分区，并在进一步读取不太可能带来帮助时停止从 SSD 读取。这显著减少了不必要的 SSD 读取和过滤工作，从而在高召回率下提升吞吐量并降低延迟。实验结果表明，KuroANN 的性能优于 PipeANN-Filter 和 Faiss，最高分别提升 34.5× 和 9.10×。

1 引言

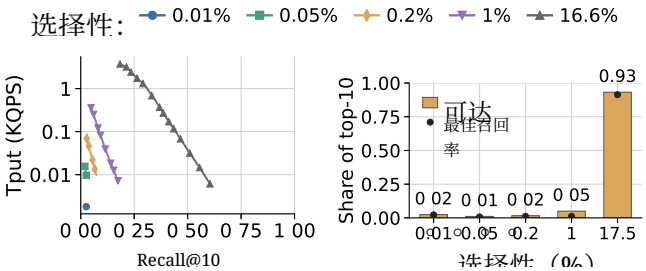
现代 AI 应用（如检索增强生成（RAG） [27], 语义搜索 [16],和推荐 [5] ）依赖于基于嵌入向量的近似最近邻（ ANN）搜索 [15, 68, 69]。这些搜索通常带有一个过滤器 [12, 22, 37, 49, 52, 56],，例如，在标记为狗的照片中寻找与查询照片最接近的照片。通过过滤器的向量所占比例称为其选择性，不同查询之间可能相差几个数量级 [9, 29, 31]。对于大型数据集，向量及其索引越来越多地存储在固态硬盘上 [10, 17, 46, 47, 61, 62],，而搜索仍必须达到较高的召回率（例如 0.99） [1, 10, 12, 38, 41]。

大多数基于固态硬盘的近似最近邻系统会构建图索引 [10,11,46, 54],，其中搜索会沿着从一个向量指向另一个向量的链接进行，并为每个访问的向量读取一条 SSD记录。不幸的是，在使用过滤器时，跳过不匹配的向量可能会使搜索无法触及匹配向量。尽管存在修复方案 [9,12],，但它们会带来额外的 SSD 读取次数或占用更多索引空间。

相比之下，分区索引无需进行此类修复，因为查询通过向量所属的分区访问每个向量。它将向量划分为多个分区，每个分区由其质心表示，而查询只扫描质心最近的分区 [7]。然而，现有的分区索引解决方案会为 选择性范围内的所有查询分配相同的搜索预算（即 *SSD* 读取次数） [56] 或搜索路径 [7, 52],，或通过固定公式设定预算 [18]。该预算决定要探测的分区数量，以及要重排序的向量候选项数量（以纠正压缩编码造成的误差）。通常，预算会设为所需预算最高的查询对应的值。与内存中 ANN 搜索不同，基于 SSD 的 ANN 搜索会产生显著开销，因为每个经过重排序的候选项都会导致一次随机 I/O 读取。因此，对所有查询采用相同的预算，可能会使 SSD 上的吞吐量降低多达一个数量级（§2）。现有解决方案还会对远多于查询实际排序数量的向量应用过滤器 [7, 18, 52, 56]。例如，位图过滤会标记数据集中的每个匹配项，即使查询只探测少数几个分区也不例外，从而增加额外的 CPU 开销。在此背景下，我们提出以下问题：*SSD* 上的向量搜索能否在保持 *SSD* 读取次数和过滤工作量较低的同时，为任意选择性的过滤器实现高召回率？

为肯定地回答这一问题，我们提出了 KuroANN，这是一种面向固态硬盘的过滤向量搜索系统，可在保持高召回率的同时减少 SSD 读取次数。KuroANN 根据满足查询过滤器的向量数量为每个查询设定读取预算，我们将其称为匹配数量。我们的关键洞见是，令人意外的是，即使查询尚未发出任何读取请求，匹配数量也能预测该查询为达到高召回率所需的 SSD 读取次数。这是因为：1）查询根据内存中的压缩编码对匹配项排序，并且只从固态硬盘读取排名靠前的候选项来计算其精确距离；2）这些候选项中排序错误的匹配项数量取决于匹配数量。

KuroANN 中的查询会经过四个步骤，其中三个步骤应用了这一洞见。首先，它会按照每个标签的匹配项在内存中压缩编码所在的位置，对这些匹配项进行排序（§3.2）。该列表的长度给出了搜索前的计数，而两次二分搜索则给出了每个探测分区中的匹配项。其次，它从一个计划中获取其计数设置，



(a) SIFT-1B 上的 PipeANN-Post

(b) LAION- 上的 Filtered-DiskANN 100M

图 1：在固态硬盘上对图进行过滤。(a) 后过滤会用不匹配向量填满其候选项列表，因此召回率一直很低。(b) 对于选择性标签而言，从其标签的入口向量出发能够到达的真实近邻很少，因此召回率接近于零。

该表记录了离线测得的、用于满足目标召回率的最快设置（§3.3）。第三，它会在未读取的候选项不太可能击败迄今为止读取到的前 k 个最佳结果时停止读取，这一判断使用了按匹配数量记录的压缩误差（§3.5）。剩余步骤会对匹配项的压缩编码进行排序，并按照有序列表的顺序预取这些编码（§3.4）。

我们实现了 KuroANN，并在各种数据集上将其与现有的基于图和基于分区的解决方案进行对比评估。实验结果证明了 KuroANN 的效率。例如，在 1 亿个向量上达到相同召回率时，KuroANN 相比 PipeANN-Filter [12] 和 Faiss [7] 的性能最高分别提升 34。5× 和 9。10×

总而言之，本文作出以下贡献：

• 我们证明，在分区索引中，匹配数量可以预测查询所需的 SSD 读取次数以及其压缩编码的误差。

• 我们设计了 KuroANN，根据查询的匹配数量设定其读取次数，从而避免在基于 SSD 的过滤向量搜索中进行不必要的 SSD 读取。

• 我们证明，KuroANN 在高召回率下优于基于图和基于分区的系统。

2 背景与动机

2.1 固态硬盘上的向量搜索

图索引和分区索引是近似最近邻搜索的两大主流索引系列。这里，我们将介绍每种索引系列的工作原理，以及它们如何在固态硬盘上处理查询。

图索引。 图索引将每个向量与数十个邻近向量相连 [46]。在固态硬盘上，它会将每个原始向量及其链接存储在同一条固态硬盘记录中，并在内存中保存每个向量的简短压缩码 [10,46,54]。乘积量化 (PQ) [20] 生成这种编码，称为*PQ* 码。搜索从一个入口向量开始，逐步跳转到与查询向量越来越接近的向量。每次跳转时，它都会读取最近的未访问候选项的记录，按 PQ 编码对其近邻排序，并根据原始向量对已访问的向量重新排序。因此，每次搜索都会发出一连串小型依赖读取，如同在固态硬盘上的其他遍历一样 [70,71]。

分区索引。 分区索引（也称倒排文件索引，IVF）通过围绕 k 均值质心 [7] 将向量划分为多个分区。查询会将其向量与质心进行比较，并探测最近的分区。在内存中，它会比较这些分区中的原始向量，或对其 PQ 编码进行排序。使用 PQ 编码时，它还可以用原始向量对其最佳候选项候选池 进行重排序。其预算（例如探测分区的数量和候选池大小）决定了搜索范围以及重排序的候选项数量；预算越大，召回率越高，但成本也越高。在固态硬盘上，PQ 编码保留在内存中，而原始向量存储在固态硬盘上 [48,64,65]。因此，查询必须对候选池进行重排序，每个候选项都需要一次随机读取。在目标召回率较高时，一条查询会读取数百个甚至更多候选项 [60]。

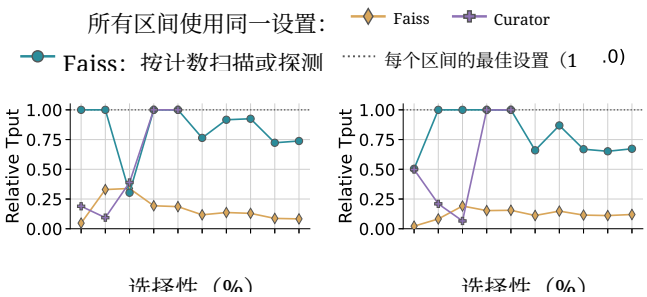
2.2 固态硬盘上过滤搜索的局限性

过滤向量搜索。 在生产环境中，查询通常带有谓词，例如标签谓词，要求查询包含所有标签（与）或至少一个标签（或） [9, 12]。过滤向量搜索会在满足查询谓词的匹配项（即向量）中，返回距离查询最近的k 个向量 [29, 37]。匹配项的数量称为匹配数量，简称计数；匹配项占全部向量的比例称为选择性。例如，标签为猫的查询中，1 亿个向量里有 1% 带有该标签，因此选择性为 1%，计数为 100 万。我们的实验将单标签查询按选择性划分为多个区间，该查询属于 1% 区间。下面将说明，为什么现有的 SSD 上向量搜索在此类过滤条件下表现不佳。

图索引在过滤条件下会失去连通性。过滤图搜索的一种常见方法是后过滤，即读取向量并丢弃不匹配的向量，从而浪费 SSD 读取次数。我们用 PipeANN-Post 表示 PipeANN [10]，这一较新的基于 SSD 的图系统，在其后过滤路径 [12]被强制启用的情况下运行。图1 （a）显示，在每个频带下，其召回率都低于 0.9，而吞吐量则下降了数个数量级。在遍历过程中跳过不匹配的向量可以避免无谓的读取，但这会截断图的路径，使许多匹配项变得不可达，如图3（a）所示。

基于 SSD 的图系统通过两种方式修复这些路径，但两者代价都很高。推测式过滤在 PipeANN-Filter 中 [12]使用内存中的布隆过滤器遍历匹配项的超集，其中的假阳性会带来 SSD 读取次数开销；同时，它为每条记录增加 2 跳邻居，我们预计这会使十亿级索引的规模扩大十倍以上。标签感知索引在 Filtered-DiskANN 中 [9]利用标签构建图，并为每个标签设置一个入口向量；其搜索仅沿着携带查询标签的近邻进行。在选择性标签下，剪枝会使大多数匹配项与其标签的入口向量断开连接，召回率因此接近于零，如图1（b）所示。

分区索引会过度配置，在固态硬盘上代价高昂。许多已过滤搜索系统都基于分区索引构建 [7,22,52,56]。与图索引不同，分区索引不会因过滤器而失效，因为查询无需经过



（a）目标召回率 0.9　（b）目标召回率 0.99

图 2：对所有选择性区间使用单一设置，其吞吐量仅达到每个区间最佳设置的一小部分（LAION-100M）。我们仅展示 Curator 在 2% 以内的结果，因为它针对低选择性 [22]。

表1：与现有设计的比较。

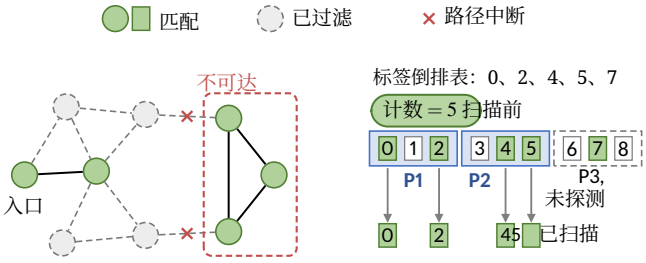
	图 [9, 10, 12]	分区 [7, 22, 52, 56]	KuroANN
每个查询集的读取次数由……设定	图遍历	选择性范围	查询的匹配数量
对任何过滤器均实现高召回率	△	✓	✓
低 SSD 输入输出	✗	✗	✓
低过滤成本	✗	✗	✓

△：在我们的测试中，PipeANN-Filter 只有在强制扫描所有匹配项时才有效；对于选择性标签，Filtered-DiskANN 则不行。PipeANN-Filter 根据选择性估计值设定候选池大小，而 Filtered-DiskANN 和 Curator（单个标签）过滤开销很低。

不匹配的向量即可触达匹配项。然而，其搜索预算（即要探测的分区数量以及要重排序的候选项数量）不会随每个查询的匹配计数而变化。例如，Faiss [7]对每个查询都使用相同的探测数量和重排序候选池大小。当原始向量位于内存中时，额外的候选项几乎不会增加成本；但在固态硬盘上，每个额外的候选项都会带来一次随机读取 [60]。

由于基于分区的片上近似最近邻系统 [3, 60] 不支持过滤，我们扩展了 Faiss 和 Curator [22]，使其支持固态硬盘，如 §2.1 所述。对于 Faiss，我们将用于重排序的原始向量移至固态硬盘。对于 Curator，我们在分区搜索期间用乘积量化估计替代原始向量距离，并在其固定容量列表中使用固态硬盘上的原始向量对最终候选项进行重排序。图 2 展示了它们在各选择性区间的吞吐量。在某些区间，吞吐量不到最佳设置下的十分之一，因为所有区间都必须采用同一设置，以便为最难处理的查询提供足够的读取次数。正如 Faiss 作者所建议的，匹配项较少的查询可以扫描所有匹配项（即穷举路径），而不必探测分区（即分区路径）；根据计数选择路径，可以挽回大部分性能损失。即便如此，同一路径中的所有查询仍使用同一个重排序候选池，其大小按所需读取次数最多的查询设定，因此较简单的查询仍会浪费 SSD 读取次数。同样，根据选择性设定的预算比按计数设定更贴合需求，但对于其范围内所需读取次数最多的查询，预算仍然过度配置 [56]，或者遵循固定公式 [18]。

与此同时，这些系统还会因以下方式产生 CPU 开销：



（a）图索引　（b）分区索引

图3：过滤影响示例。（a）移除图节点会破坏通往匹配向量的路径。（b）分区探测仅对其分区中匹配的 PQ 码进行排序。

过滤的向量数量超过查询所需，进一步降低吞吐量。即使查询只探测少数几个分区，位图过滤也会标记查询的每个匹配项 [18, 52, 56]，而内联过滤会测试每个被探测的向量 [7]。每标签列表 [22, 25] 可避免单个标签带来的这项开销，但对于 AND谓词，它们会先计算出所有匹配项（例如，通过求标签列表的交集）。表 1 对比了这些设计。

2.3 我们的方法

我们的总体思路是根据查询在分区索引上的匹配数量，设定每个查询的 SSD 读取次数。

为什么匹配数量决定读取次数。匹配数量表明查询必须读取多少候选项。查询根据 PQ 码对候选项进行排序；PQ 码的压缩误差可能会让某些候选项看起来比真实近邻更接近查询。当匹配向量更多时，这类候选项可能也会更多，因此查询往往需要读取更多候选项，才能达到相同的召回率。这解释了为什么图2 中的最佳设置会随频带变化。然而，很难用公式预测给定匹配数量对应的读取次数。这是因为固定设置的召回率会随匹配数量变化，并且在分区路径和穷举路径上呈相反趋势（§3.3）。

查找计数和匹配项。 分区索引可以在任何 SSD 读取发生之前给出这个计数。图 3 (b) 展示了一个包含三个分区、共九个向量的示例。它们的 PQ 编码按分区顺序存储在内存中，编号为 0 到 8。对于每个标签，我们都会保存一个有序列表，列出带有该标签的向量编号，这称为该标签的 倒排列表。在这个示例中，该标签的倒排列表为 0、2、4、5、7。其长度为 5，因此无需进行任何 SSD 读取即可得出计数。此外，由于每个分区包含连续编号，倒排列表还显示了各分区中的匹配项。因此，探测 P1 和 P2 时，只需对 PQ 码 0、2、4 和 5 排序，无需检查其他向量。

测量设置与校准停止点。 我们在几个计数下测量满足目标的最快设置，并将这些设置保存在一个小表中。此外，我们还测量每个计数下 PQ 编码的误差；一旦剩余未读取的候选项不太可能包含真正的近邻，查询就会停止读取。因此，与使用单一设置相比，每个查询读取的向量要少得多。

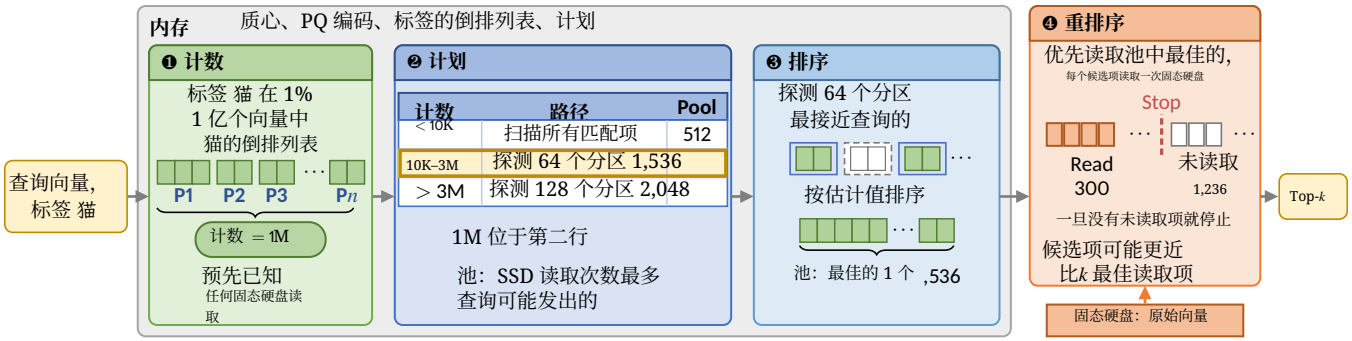


图 4：KuroANN 流水线示例，展示了一个查询标签为猫的查询；1% 的 1 亿个向量带有该标签。只有重排序会读取固态硬盘，并且会在读取完池中的内容之前很久就停止。

3 KuroANN设计

3.1 概述

机制。 KuroANN通过四个步骤处理每个查询，图 4 展示了其中一个查询的处理过程。首先，它从倒排列表中获取查询的计数和匹配项，然后根据该计数从计划中选择一个设置。接下来，它对内存中的匹配项的 PQ 编码进行排序，最后通过固态硬盘读取对最佳候选项进行重排序，直到可以停止为止。所有向量的 PQ 编码都位于一个按分区顺序排列的数组中，而固态硬盘保存着原始向量。

- **按位置排序的倒排列表（§3.2）。** 我们将数组中 PQ 码的索引称为其位置。每个标签的倒排列表列出携带该标签的向量的已排序位置。其长度给出计数，而两次二分搜索即可得到每个探测分区中的匹配项。属性值和向量标识符存放在按位置索引的数组中。
- **按计数索引的计划（§3.3）。** 该计划是一个小型表格，其行覆盖不同的计数范围。每一行确定路径、预算以及何时停止读取，并通过离线校准测量每一行的召回率。
- **前瞻扫描（§3.4）。** 扫描会对匹配的 PQ 编码进行排序，并在计算当前候选项的同时，获取后续候选项的编码。
- **校准停止（§3.5）。** 重排序会从固态硬盘读取最优候选项，每个候选项执行一次对齐读取；当误差界限达到查询的计数时停止。

示例。 图 4 展示了对标签 *cat*的一次查询；该标签由 1 亿个向量中的 1% 携带。标签 *cat*的倒排列表长度给出计数 100 万（❶）。这个计数对应计划中的第二行，该行以包含 1,536 个候选项的池探测最近的 64 个分区（❷）。探测会在内存中对这些分区内匹配项的 PQ 编码进行排序，并保留最优的 1,536 个（❸）。最后，重排序按排序顺序从固态硬盘读取这些候选项；当所有尚未读取的候选项都不太可能比目前已读取的 k 个最佳项更近时，就会停止（❹）。在此示例中，读取 300 次后便停止，KuroANN 返回已读取的 k 个最近向量。

3.2 位置顺序倒排列表上的谓词

在第一步中，查询会先获取其计数以及所探测的每个分区中的匹配项，然后才计算任何距离。每个标签的倒排列表长度可给出计数，但只列出向量标识符的倒排列表无法指出哪些匹配项位于探测分区中。因此，探测必须在位图中标记所有匹配项，或检查所访问的每个 PQ 码的标签；即使查询只探测少数几个分区，这也会造成浪费。KuroANN 则将每个标签的倒排列表按位置排序（即按其匹配项的 PQ 编码在内存中的位置排序）。由于同一分区的 PQ 编码彼此相邻，该分区的匹配项会在倒排列表中构成一个连续区段。因此，只需通过两次二分搜索确定分区边界，就能找出每个探测分区的匹配项，无需为每次查询建立位图。此外，穷举路径会按位置顺序读取相同的倒排列表，因此它会顺序访问内存中的 PQ 编码，而非随机访问。

KuroANN 在每个探测分区内评估多标签谓词，而不是在整个数据集上评估。对于与谓词，它从最稀有的标签开始求交各倒排列表；对于或谓词，则合并各倒排列表并去除重复项。由于结果保持位置顺序，过滤工作只需遍历查询所探测的分区。

统计匹配项。 对于单个标签，计数就是其倒排列表的长度。对于“或”条件，计数是各倒排列表长度之和，这是一个上界，校准也会用到它。对于“与”条件，精确计数需要对倒排列表求交集，耗时会随最短倒排列表的长度增长。不过，KuroANN 只用计数来选择计划中的一行；该表的每一行都覆盖一个计数范围，因此计数无需精确。所以，当“与”条件中最短的倒排列表较长时，KuroANN 会改用置信区间估计计数。它以等间距对该倒排列表进行采样，并统计同时带有其他标签的样本数。例如，如果从包含 100 万个条目的倒排列表中抽取的 1.6 万个样本里，有 10% 匹配，那么置信区间会将计数范围确定在 9.3 万到 10.7 万之间。只有当该区间跨越多行时，KuroANN 才会对倒排列表进行精确求交。

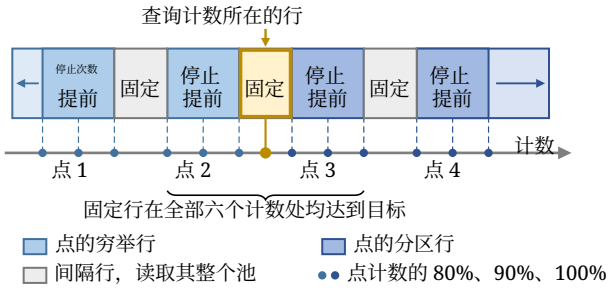


图5：按计数索引的计划示例。校准在四个计数（点1至点4）处各测量一行，并为它们之间的每个间隔测量一行固定行。查询会选取覆盖其计数的行。

其他谓词。 相同的分区内评估方式也适用于数值范围、字符串条件、否定和嵌套。在探测分区内，KuroANN 会从操作数（例如标签或范围）中选取候选项数量最少的一项。因此，如果谓词包含标签，候选项就来自该标签的倒排列表；否则，候选项来自分区中的每个位置。随后，KuroANN 按位置顺序逐一检查每个候选项是否符合整个谓词。在整个数据集范围内（例如通过穷举路径），选择性较强的范围或字符串条件会从按值排序的列副本中选取候选项。KuroANN 根据谓词各部分的计数来界定该谓词的计数范围。只有当这些界限会选出不同的计划行时，KuroANN 才会像上文那样先通过采样，再进行精确计数。

3.3 按计数索引的计划

在第二步中，查询会在其计数下选择满足目标召回率的最低成本路径和预算。成本模型可以为每个计数选择一种设置，但我们测试过的公式都无法预测过滤设置在固态硬盘上的召回率（§5.4）。因此，KuroANN 从计划中读取这些设置。计划是一张小表，列出了在少数几个计数下离线测得的、满足目标召回率的最快设置。这几个计数已足够，因为一种设置若在两个计数下都满足目标，也适用于它们之间的计数。这是因为，对于匹配项分布相似的标签，随着计数变化，固定设置在每条路径上的召回率只会朝一个方向变化。随着计数增大，穷举路径上的召回率会下降；随着计数减小，分区路径上的召回率会下降。

计划。计划的每一行都覆盖一个计数范围，并固定路径、探测分区的数量以及候选池大小。此外，对于提前停止的行，还会固定 §3.5 中的停止表及其分位数水平。在查询时，KuroANN 会选择计数所在范围对应的行，只需进行几次比较。该计划类似于数据库优化器中的参数化计划，但它以首次读取前倒排列表给出的计数作为索引。与参数化计划不同，每一行都必须满足目标召回率，而不只是将成本降至最低；KuroANN 会测量每一行的召回率，而不是对其进行估算。

KuroANN 根据留出查询离线构建计划在少数几个计数处，我们将其称为校准点。对于每个校准点和目标，系统会测量两条路径上的候选设置，并选出满足目标且裕量充足的最快设置（图5）。裕量的计算方式是从测得的召回率中减去两个标准误，并将每个校准查询的每个近邻都视为一次试验。裕量可保护计划免受有限校准查询集所带来的抽样误差影响。例如，对于包含 500 个校准查询、每个查询有 10 个近邻的情况，目标为 0.99 的行测得的召回率必须至少达到 0.993。

覆盖一系列计数。 当某个校准点的校准查询共享同一个标签时，它们只能覆盖一个计数。因此，KuroANN 还会在该标签倒排列表的副本上测量每种设置：这些副本分别保留其成员的随机 90% 和 80%，使该点的行覆盖这三个计数的范围。在这些副本中，真实近邻会根据保留的成员重新计算，因为被移除的向量不再匹配。由于这些副本会随机移除成员，和随机删除一样，因此即使删除后某个标签的计数下降，该行仍然适用。

在两个校准点之间，任一校准点对应的行都可能在另一点附近漏掉目标。这是因为召回率在两条路径上的变化方向相反。对于这样的间隙，KuroANN 会使用在这两个点的六个校准计数处都满足目标、且速度最快的固定设置，并且不会提前停止。因此，在整个间隙内都适用该设置。

示例。图5展示了一个包含四个校准点的计划。在每个点，校准会测量计数的 80%、90% 和 100% 处的设置，并在计数轴上以圆点标出。点 1 和点 2 的匹配项较少，其对应的行会扫描所有匹配项。点 3 和点 4 的匹配项较多，其对应的行会探测分区。每个点对应的行都会使用该点的停止表提前停止。在点 2 和点 3 之间，查询会采用该间隙的固定行（大括号所示），并读取整个池。同样，点 1 以下的查询会采用点 1 的行，点 4 以上的查询会采用点 4 的行。

局部计数。计数表示有多少向量匹配，但并不表示它们在向量空间中距离查询有多近。例如，当查询是一张猫的照片，且其过滤器是标签 cat 时，许多标记为 cat 的向量都靠近该查询。于是，与该计数的校准查询周围相比，这些匹配项在查询周围分布得更密集，该计数的停止表可能会过早结束读取。因此，KuroANN 还可以统计查询最近分区中的匹配项。如果该标签在这些分区中的密度远高于整体密度，KuroANN 就会使用该标签在这种密度遍布所有位置时对应的计数行，而无需重新校准。例如，某个标签包含全部向量的 1%，但包含最近分区中向量的 5%，那么它会使用计数大五倍所对应的行。

查找最近的质心。 由于当质心数量众多且维度很高时，扫描所有质心会主导分区路径，KuroANN 在质心上搜索图索引，而任何过滤器都不会破坏该索引。对于几千个质心，则采用精确扫描。构建完成后，它会将入口点无法到达的每个质心连接到一个附近的可到达质心

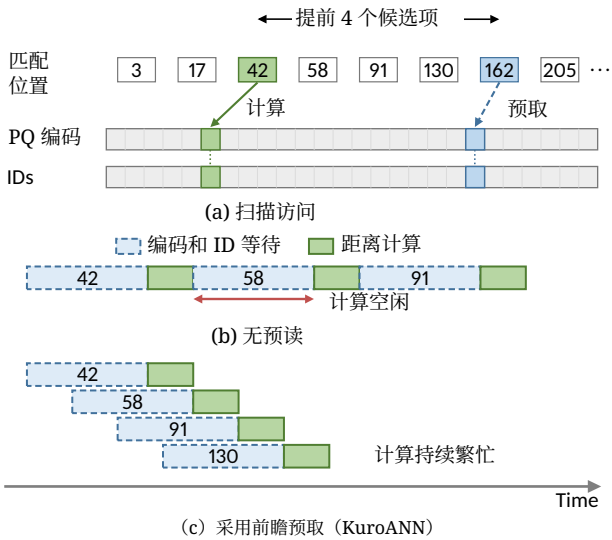


图 6：前瞻预取示例。（a）扫描计算位置 42 处的候选项时，已经预取了位置 162 处的 PQ 码和标识符。（b）不采用前瞻预取时，CPU 在每个候选项处都要等待内存。（c）采用前瞻预取后，这些等待可以与计算重叠。

，否则其所属分区将永远不会被探测。

3.4 采用前瞻预取进行扫描

在第三步中，查询会在内存中对其匹配项的 PQ 编码进行排序。借助倒排列表，过滤器只需访问匹配项，因此当读取量减少后，这一排序过程便成为内存中的主要工作，并限制吞吐量。因此，KuroANN 会在计算当前候选项的同时，预取后续候选项所需的内存，并使用大页支持扫描数组。之所以可行，是因为倒排列表预先记录了每个候选项的位置，因此扫描过程能够获知未来的访问位置。如果没有这种前瞻预取，普通循环几乎会在每个候选项处等待内存，因为匹配位置虽已排序，却十分稀疏，连续的候选项很少共享同一个缓存行。它们的 PQ 编码和标识符还位于数 GB 大小的数组中，使用常规页时超出了 TLB 的覆盖范围。

前瞻预取。 KuroANN 以固定的前瞻距离交错访问多个候选项的内存。在图 6中，前瞻距离为四个候选项，覆盖位置 3、17、42、58 等匹配位置。请注意，交错执行彼此独立的内存访问是隐藏内存延迟的已知方法。前瞻距离是一个机器常量，只需设定一次，即可覆盖一次 DRAM 往返所需的时间。大页通过减少这些大型数组上的 TLB 漏检来补充前瞻机制。由于这两种机制只会改变请求内存的时机和页面的映射方式，因此启用或禁用它们时，候选项和结果都完全相同。

3.5 在有界 SSD 读取次数下进行重排序

最后一步，重排序会从固态硬盘读取最佳候选项的原始向量，并返回精确近邻。在

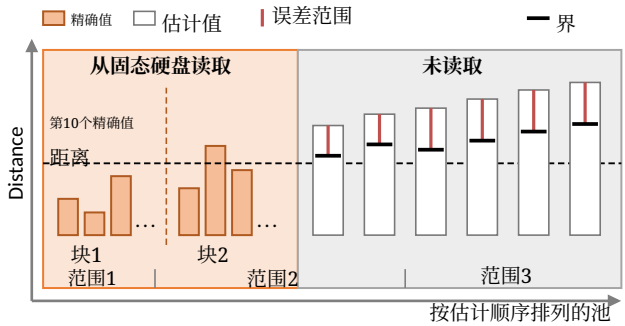


图 7：10 个近邻的校准停止示例。每个尚未读取的候选项都有一个界（黑色标记），其估计值会减去根据其排序测得的误差。当所有界都高于目前已读取的第 10 个最佳精确距离（虚线）时，读取便会停止，因为此时尚未读取的候选项不太可能更近。

高召回率场景下，这些读取会占据查询时间的主要部分，因此 KuroANN 通过两种方式对其进行界定。它将每个向量存储在按块对齐的槽位中，因此每次读取都会获取一个完整向量。它还会在根据其计数测得的误差界限下终止每个查询，因此大多数查询都能提前停止。相比之下，一个跨越两个 SSD 块存储的向量需要读取两次，而固定池在答案已经确定后仍会继续读取。一旦下一个估计值超过当前第 k 个最佳精确距离就停止同样并不安全，因为 PQ 估计值可能超过精确距离，真正的近邻随后可能位于停止点之后。

每个候选项只进行一次对齐读取。例如，将 3,072 字节的向量连续打包到 4KB 块中时，其中一半会跨越块边界。此时，直接 I/O 必须读取两个块。为避免这种情况，块对齐槽位会在每个块中容纳尽可能多的完整向量；根据候选项的标识符，通过一次除法即可确定其所在块。

校准停止。 多步搜索通过已证明的距离下界，避免漏掉近邻，但仅凭 PQ 编码无法得到这样的下界。在校准停止机制下，KuroANN 会在校准查询中，测量候选池每个排名上所有估计值的误差。误差等于精确距离减去估计值。对于每个候选池排名范围，KuroANN 会在停止表中保留这些误差的一个低分位数。该分位数通常为负数。查询时，每个尚未读取的候选项都会获得一个乐观界，即其估计值加上对应排名范围的分位数。系统按数据块发起读取，并在第k个最优精确距离不大于最小的未读取下界时停止。

示例。图 7 展示了针对查询 10 个近邻的情形。KuroANN 按估计顺序读取池中的候选项，每次读取一个块。每读取完一个块，已读取候选项的距离便是精确距离（橙色条），其中第 10 小的距离确定了虚线的位置。KuroANN 将每个未读取候选项（白色条）的估计值下调其排序范围的误差范围，从而得到其界（黑色标记）。读取块2后，所有未读取候选项的界都高于虚线，因此 KuroANN 无需读取池中的其余候选项便停止。

每个计数对应一张表。给定排序位置上的估计误差取决于计数。因此，一张供所有计数使用的表对某些计数而言不够精确。于是，我们的设计为每个校准点和路径各保留一张表。每个提前停止的计划行都会使用其对应校准点的表，并采用由校准过程为该行选择的分位数水平。需要注意的是，已知采样误差分位数可以剪枝距离计算；而在这里，每个计数对应的一张表决定何时停止 SSD 读取。不过，分位数描述的是单个候选项，而未读取集合取决于相同的估计值。因此，校准停止是一种启发式方法，无法保证对每个查询都有效。
</output>

3.6 更新

插入和删除应立即生效，无需重建索引或重新校准计划。然而，PQ 编码和倒排列表是按位置顺序排列的紧凑数组，若直接在分区中插入向量，就会导致之后的所有位置都发生偏移。所幸，计划只读取计数，而不读取位置。因此，我们会及时更新计数，但在合并之前保持紧凑数组不变。插入操作会写入增量段，删除操作会设置墓碑标记，并通过偶尔进行的合并来重写紧凑数组。

增量段与合并。KuroANN 会将插入项添加到每个分区各自的增量段中，其中存储新的 PQ 编码、标识符和标签。由于增量段不存储属性值，目前的更新仅支持标签谓词。添加一个新向量需要进行一次最近质心搜索和一次编码，因为编码器会复用训练后的质心和 PQ 码本。搜索期间，分区路径会在扫描完探测分区中的紧凑条目后，扫描其中的增量条目。对于穷举路径，每个标签都会保留其增量条目的引用。

删除会设置一个墓碑位，扫描时会跳过已标记墓碑的向量。这两种操作都会更新每个标签的计数，计划会读取这些计数。因此，随机插入和删除无需重新校准。有待处理的更新时，与操作照常在紧凑数组上计数，其匹配的增量条目也会计入计数。按需合并时，会将增量段移入紧凑数组，并丢弃已标记墓碑的向量。它会按标识符升序写入每个分区，这与重新构建时的顺序一致。因此，插入留出向量后再进行合并，得到的数组与使用这些向量构建的索引相同，但距离两个质心同样近的向量除外。

3.7 讨论

校准成本与工作负载漂移。 KuroANN 特有的成本是计划校准，因为扫描探测数量和候选池大小是分区索引的标准做法。校准需要使用若干计数下的留出查询及其精确过滤结果，每种工作负载只需运行一次，并会为每个目标召回率生成一个仅有几千字节的计划。不过，若要采用新的目标召回率、重新训练质心或码书，或者查询与标签的关联方式不同于校准查询，就需要重新校准计划。当

表 2：评估中使用的数据集。

数据集	向量	Dim. Type	标签	每个向量的标签数
LAION-100M	100M	768 个浮点数	10,824	平均 5.0
SIFT-1B	1B	128 无符号8位整数	43,592	平均值为 6.0
YFCC-10M	10M	192 无符号8位整数	200,386	平均值为 10.8
YT-5M	5M	1024 浮点数	3,862	平均 3.0
PubMed	23.9M	1024 浮点数	29,912	12.0 平均值
Amazon	35.1M	768 浮点数	12,996	平均 6.0

查询组合发生漂移时，操作员可以根据已记录查询离线重建计划，并替换旧计划。

计数校准读取适用的场景。这一原则要求索引具备两个属性。首先，索引必须在执行高开销工作之前计算出查询的计数，或者至少将其限定在一个计划行内。其次，在每条路径上，固定设置的召回率必须随计数朝同一方向变化。第二个属性是通过观察得出的，并非必然成立；当某个标签簇的匹配项聚集在查询附近时，该属性便不成立，而局部计数可以检测到这种情况。对于能够在内存中枚举匹配项的任何谓词，分区索引都具备第一个属性。对于匹配项分布方式类似于校准标签的标签，分区索引具备第二个属性。图索引则不具备这一属性，因为其召回率取决于匹配项之间的路径。由于这两个属性都与固态硬盘无关，我们预计这一原则也适用于内存分区索引。

4 实现

我们用约 1.5K 行 C 代码实现了 KuroANN 搜索引擎++ ，基于 `liburing`。Python 和 C++ 工具用于构建索引并校准计划。构建器使用 Faiss 训练质心和 PQ 码本 [7]。每个 PQ 码的每个片段占一个字节，因此 LAION-100M 和 YFCC-10M 上为 64 字节，SIFT-1B 上为 32 字节，YT-5M 上为 128 字节。分区数取最接近每 2,500 个向量一个分区的 2 的幂，最多为 32,768。一个覆盖质心的 HNSW 图 [35] 用于查找最近分区。扫描会提前预取八个候选项 [4]，，PQ 编码和倒排列表由透明大页提供支持 [24]。每个搜索线程都通过自己的 `io_uring` 环 [23]提交读取。固定计划行会一次性提交整个池，以保持大量读取处于待处理状态 [13]。相比之下，采用校准停止规则的计划行每次提交 32 次读取，并在每批之后检查停止规则。

5 评估

5.1 方法

测试平台。我们的服务器配备两颗 Intel Xeon Silver 4514Y CPU，每颗有 16 个核心，256 GB DDR5 内存，以及一块 8 TB WD Black SN850X NVMe 固态硬盘。该固态硬盘每秒可处理 872K 次随机 4 KiB 读取。服务器运行 Ubuntu 24.04 和 Linux 6.17。每个系统都使用

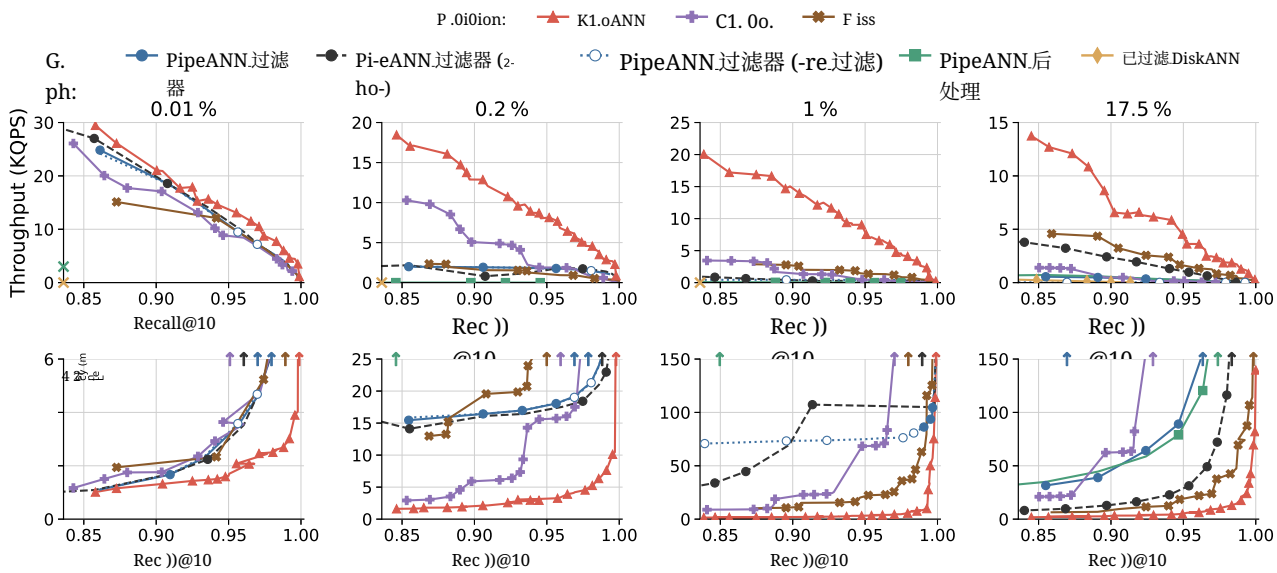


图 8：吞吐量和延迟随召回率的变化。KuroANN 全面领先所有基线，只有在选择性最低时，PipeANN-Filter 的表现较为接近。

○_直接读取，在释放页缓存后读取其固态硬盘上的文件。每个系统以 32 个线程进行闭环搜索， [42],将线程固定在 32 个物理核心上，并在两个处理器插槽之间交错分配内存。

对比系统。 我们将 KuroANN 与三个基于图的系统进行比较。PipeANN-Filter [12] 采用推测性过滤，PipeANN-Post 强制使用 Pi- peANN 的后过滤路径 [10,12], , 而 Filtered-DiskANN [9] 则将标签构建到图中。我们报告速度更快的 PipeANN-Filter 构建版本在启用和不启用 2 跳邻居时的结果，也报告强制使用其预过滤路径时的结果。不过，对于 SIFT-1B 和 §5.4中的随机标签，2 跳邻居无法装入固态硬盘。Filtered-DiskANN 运行 DiskANN 的已过滤固态硬盘工作流，并采用两项不会改变其搜索过程的崩溃修复；在带有均匀标签的 LAION-10M 上，其召回率达到 0.99。

我们还将 Curator [22]和 Faiss [7] 的内存分区索引改造为适用于固态硬盘的版本。两者都会对 KuroANN 的 PQ 编码进行排序，并从其固态硬盘存储中重排序；Faiss 还会使用其质心。在 YFCC-10M 上，改造后的 Curator 在相同搜索预算下达到了官方内存版 Curator 的召回率。计数低于阈值时，Faiss 会按照其作者的建议扫描匹配项 [7],；高于阈值时，则探测最近分区。

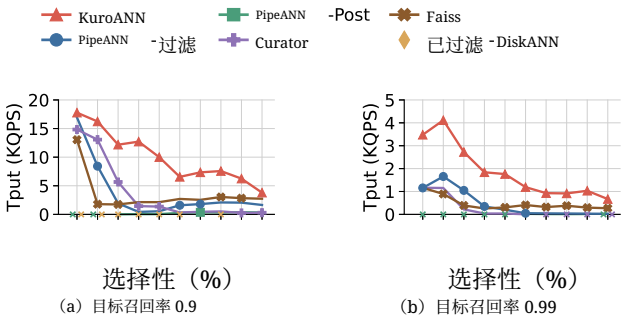
数据集。表2 列出了这六个数据集。LAION-100M [43,72]和 SIFT-1B [21] 没有可用于过滤搜索的自然标签。因此，我们为它们赋予合成标签，每个向量五个或六个，这些标签来自不同规模的 k 均值聚类。合成标签可让每个区间达到指定的选择性；而聚类生成的标签与真实类别相似，因为同一类别中的成员通常彼此相似。LAION-100M 上的区间范围为 0.01% 至 17.5%，SIFT-1B 上则为 0.01% 至 16.6%。由于这些标签会使同一标签下的向量彼此靠近，§5.4 还评估了与向量独立抽取的标签。YFCC-10M 和 YT-5M 具有自然标签。与对 PipeANN-Filter 的评估一样 [12]，我们在 YFCC-10M 上构造 AND 查询，在 YT-5M 上构造 OR 查询，并将每个数据集划分为四个区间。PubMed 和 Amazon 来自两个采用自然过滤条件的实际应用（§5.3）。

校准与比较。KuroANN 在 4.2 小时内，针对每个评估标签区间的 500 个留出查询校准其计划。随后，它在评估前固定这些计划；评估使用每个区间的另外 500 个查询。相比之下，每个基线都在评估查询上扫描参数。所有计划都在每个数据集上满足所有目标。对每个基线，读取次数按 KuroANN 的召回率计算；若没有任何设置与之匹配，则在两个设置之间插值。每个查询请求 10 个近邻，第 10 近距离上的并列项均计为已找到。吞吐量（每秒查询数，QPS）和延迟来自计时测试。

5.2 主要结果

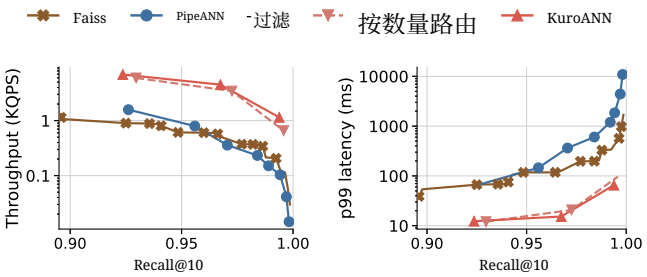
吞吐量和延迟与召回率的关系。图8 展示了 LAION-100M 在四个区间下，吞吐量和中位延迟随召回率的变化。十字标记表示数值低于图示范围，箭头标记表示数值高于图示范围。Filtered-DiskANN 仅出现在吞吐量一行，因为它没有报告中位延迟。可以看到，除 0.01% 区间外，在所有区间中，当召回率不超过 0.999 时，KuroANN 都领先于每个基线。在 0.01% 区间，PipeANN-Filter 采用其预过滤路径，只对少数匹配的 PQ 编码进行排序；在召回率低于 0.93 时，两条曲线几乎重合。在其他区间，它的图遍历会将 SSD 读取次数耗费在绕行路径上。强制采用其预过滤路径（虚线）可以避免这些读取，但会对所有匹配的 PQ 编码进行排序。因此，在 1% 区间、目标为 0.9 时，KuroANN 的中位响应速度仍比两种路径中较快的一种快 32 倍。0× 此外，在所有四个区间中，Curator 和 Faiss 的表现都不及 KuroANN。

选择性的影响。 图 9 显示，在 LAION-100M 的 0.9 和 0.99 目标召回率下，KuroANN 在每个区间都领先于所有基线方法



（a）目标召回率 0.9　　（b）目标召回率 0.99

图 9：选择性的影响。KuroANN 在每种选择性下都优于所有基线。



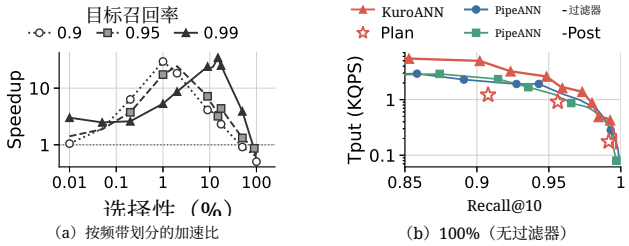
（a）吞吐量　　（b）尾延迟

图10：混合选择率的影响。对于整个数据流仅使用一个配置时，所有基线在吞吐量和尾延迟方面都远远落后于 KuroANN。

。交叉标记表示某个基线方法的召回率低于 KuroANN。在 0.99 目标下，KuroANN 的几何平均性能比 PipeANN-Filter 高出 10.0×，最高可达 34.5×；当选择性达到 9% 及以上时，领先幅度最大。这是因为 PipeANN-Filter 只有在其预过滤路径上才能达到 KuroANN 的召回率，而该路径需要为每个查询对数百万个匹配的 PQ 码进行排序。在最稀疏的三个区间中，领先幅度较小，因为 PipeANN-Filter 每个查询读取的向量数量比 KuroANN 多 2.11× 至 3.54×。在 0.9 目标下，KuroANN 的性能也比 Faiss 最高高出 9.10×，因为 Faiss 的固定池必须覆盖某个区间中需要读取最多数据的查询。与此同时，在 0.99 目标下，当选择性高于 9% 时，Curator 无法达到 KuroANN 的召回率；而在 0.9 目标下，Filtered-DiskANN 仅在三个区间达到该召回率。KuroANN 的性能比 PipeANN-Post 高出 11×或更多，这是因为过滤会使其图断连，或迫使其采用很长的绕行路径。

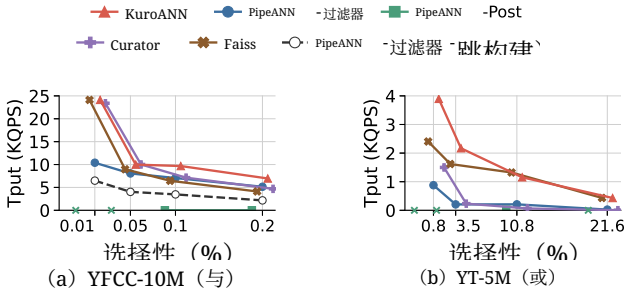
混合选择率的影响。 图 10 向每个系统发送一个包含 500 个 LAION-100M 查询的数据流，每个频带 50 个查询。每个系统都在整个数据流上运行其最佳单一配置。虚线表示使用 Faiss 的阈值规则运行 KuroANN 引擎，我们称之为按数量路由，而不是使用其计划。可以看到，KuroANN 比 Faiss 快 6。9× 至 8。2×倍，比 PipeANN-3× Filter 快 4。1× 至 11。4× 倍（召回率取其三个计划中的相应值）。此外，在相同召回率下，其 p99 延迟也比二者低 5。4×至 28。

宽谓词的影响。 大多数向量满足的谓词



（a）按频带划分的加速比　　（b）100%（无过滤器）

图11：宽谓词的影响。KuroANN 的领先优势在中等选择率时达到峰值，并在两端收窄，但在无过滤器的情况下，KuroANN 仍然领先。

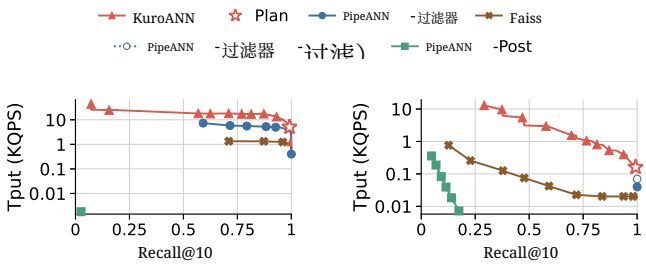


（a）YFCC-10M（与）　　（b）YT-5M（或）

图12：与和或谓词的影响。在两种情况下，KuroANN 都领先于 PipeANN-Filter，而经过调整的分区索引也已接近。

适用于未过滤搜索，此时图索引无需修复。因此，我们运行 OR 谓词，分别有 49.9% 和 89.7% 的 LAION-100M 向量满足其中一个谓词，另一个谓词则覆盖所有向量；计划采用冻结设置，其频带范围最高达 17.5%。图 11 (a) 显示，相较于 PipeANN-Filter 和 PipeANN-Post 中速度较快者，KuroANN 的领先优势在中间区间达到峰值，最高为 34。5×在 0.99 时，而在两端则有所缩小。没有过滤器时，PipeANN-Filter 的速度最多快 85× 于这些计划，因为它探测的分区比这些谓词所需的更多。不过，图 17× 11 99× 所示，KuroANN 的固定设置仍比 PipeANN-Filter 和 PipeANN-Post 中速度较快者领先 1。11 (b)（星号表示这些计划）。

与和或谓词的影响。 图 12 报告了 YFCC-10M 上的 AND 查询，这些查询匹配的向量少于 0.5%。它还报告了 YT-5M 上的或查询，这些查询匹配 0.8% 到 21.6% 的向量。虚线表示使用其 2 跳邻居的 PipeANN- Filter，叉号表示召回率低于 KuroANN 的设置。我们跳过 Filtered-DiskANN，因为其搜索每个查询只读取一个标签。在 AND 查询上，KuroANN 领先 PipeANN-Filter 1。24× 到 2。32×，尽管 PipeANN-Filter 在那里也进行了预过滤。这是因为 PipeANN-Filter 从固态硬盘读取标签列表，并为每个重排序的候选项读取两个 4 KiB 扇区。相比之下，KuroANN 在内存中求交倒排列表，并对 16 到 32 个候选项进行重排序。由于这些计数很小，Curator 和 Faiss 也只对少量 PQ 编码进行排序，并且在两个最稀疏的区间中，二者之一分别与 KuroANN 持平。在或查询上，KuroANN 最多领先 PipeANN-Filter 17。3×。然而，Faiss 这一



(a) 0.01% (b) 1%

图13：10亿数据集（SIFT-1B）上的吞吐量与召回率。KuroANN 在 10 亿个向量的规模下仍保持领先，而在选择性最低时，PipeANN-Filter 的表现最接近。星号标出目标为 0.99 的计划。

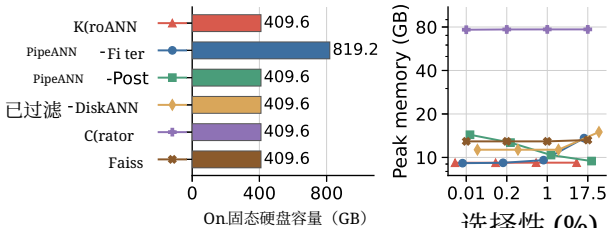
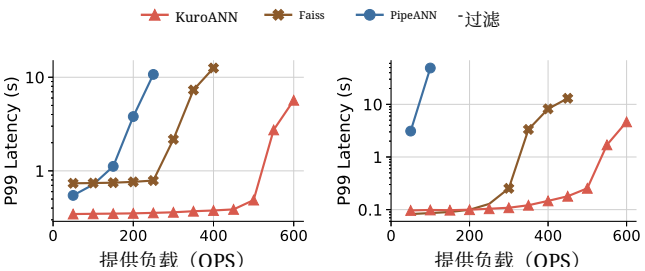


图14：固态硬盘和内存占用。与使用 2 跳邻居的 PipeANN-Filter 相比，KuroANN 存储的数据少得多；除最稀疏的两个区间外，KuroANN 的内存占用最低，而在这两个区间中，PipeANN-Filter 的内存占用与之相近。

最强基线在四个区间中的两个区间最多快 1.15× 倍，而在另外两个区间最多慢 1.62× 倍。在恰好达到 0.99 目标、而不是 KuroANN 的召回率时，Faiss 在四个区间中的三个区间更快。

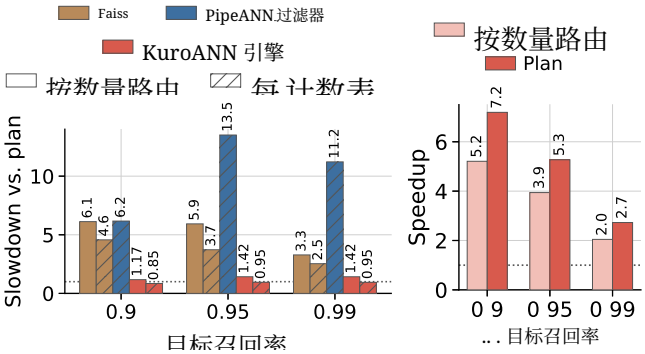
数据集规模的影响。图13 比较了 SIFT-1B 在 0.01% 和 1% 区间的召回率—吞吐量曲线。我们省略了 Curator（其内存占用会超出限制）和已过滤的 DiskANN（在 LAION-100M 的这些区间中，召回率低于 0.03）。我们采用 PipeANN-Filter 速度最快的路径，即在两个区间都进行预过滤，并且能够达到每个目标。在 0.01% 时，PipeANN-Filter 会自行进行预过滤，因此强制路径只出现在 1% 时。在 0.99 目标下，PipeANN-Filter 在 0.01% 时表现最接近，速度慢了 1。48×倍，因为只有 10 万个向量符合条件，预过滤会直接搜索这些向量。在 1% 时，KuroANN 快 2。11×倍，而 Faiss 慢 4。6× 到 8。0×倍。

固态硬盘与内存占用。图 14 展示了 LAION-100M 上各系统的固态硬盘存储大小和峰值内存。KuroANN 使用一个 409.6GB 的存储库为所有区间提供服务，其大小是 PipeANN-Filter（含 2 跳邻居）的 819.2 GB 索引的一半。在内存方面，KuroANN 的占用最多为 9.2 GB，与 PipeANN-Filter 在 0.01% 和 0.2% 区间的占用大致相当，低于其在较宽区间的占用，也低于 Filtered-DiskANN 在所有区间的占用，因为这些系统都为每个向量保留了 64 字节的 PQ 编码。其倒排列表和位置到标识符数组每个向量仅额外占用 24 字节。相比之下，Faiss 的内存占用最多为 13.2 GB，因为它还保留了每个向量的标签，以及从每个向量到其 PQ 码的映射表。Curator



(a) PubMed 上的 RAG检索 (b) Amazon 上的推荐

图15：负载下的应用。基线达到饱和时，KuroANN 仍能在高负载下保持较低的尾延迟。



(a) 针对该计划的策略 (b) 位图上的倒排列表，不进行预取

图16：优势从何而来。内存中的处理速度更快，而且读取次数会随着计数的增加而成倍增长。

占用 77.0GB，因为每个树节点都带有一个包含所有标签的布隆过滤器。

5.3 应用

现在我们展示，当查询以不断增加的速率到达时，KuroANN 如何加速两个过滤向量搜索的实际应用。这些实验在第二台服务器上运行，该服务器配备 28 核 Intel Xeon Gold 5512U CPU、128GB 内存和一块 800 GB 的 Dell DC NVMe 7450 SSD；每个系统都使用 28 个线程进行搜索。每个系统以泊松到达的方式接收某个工作负载的评估查询，并报告从到达到完成、以 0.99 为目标时的 p99 延迟。当 p99 延迟首次超过 10 秒时，曲线即结束。KuroANN 运行其 0.99 计划；对于每个频带，每个基线都采用能够达到 KuroANN 召回率的最快设置；若没有这样的设置，则采用其最准确的设置。这种选择有利于基线。

PubMed 上的 RAG 检索。 RAG 系统会检索与问题最接近的摘要，通常限定在特定范围内，例如某种出版物类型或若干年份。PubMed [40] 存储了 2,390 万篇摘要的 BGE-M3 [2] 嵌入向量，并按出版物类型、MeSH 术语和年份进行过滤；其查询内容为 bioRxiv 标题。图 15(a) 显示，KuroANN 的 p99 延迟在每秒查询数达到 500 之前始终低于 0.5 秒，而 Faiss 在每秒查询数为 274 时达到饱和，PipeANN-Filter 则在每秒查询数为 180 时达到饱和。这

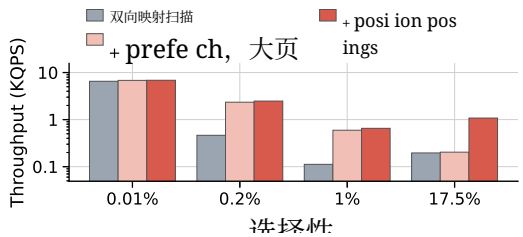


图 17：在 LAION-100M 上采用一种固定设置（探测 512 个分区，池大小为 128）对 PQ 编码扫描进行累积消融。每一步返回的结果都相同。

是因为 KuroANN 处理该数据流的速度比 Faiss 和 Pipe ANN-Filter 快 1.6× 和2.4× 倍，而且只有当请求负载超过系统的吞吐量时，才会形成队列。

Amazon 上的推荐。商店会根据购物者选定的类别、评分和价格，推荐最接近其已购买商品的商品。Amazon 拥有 Amazon Reviews'23 中 3510 万件商品的 BLaIR [14] 嵌入向量，这些向量按类别、评分和价格进行过滤；每个查询则是购物者近期购买商品的均值。图15 (b) 显示，KuroANN 可提供 492 每秒查询数，p99 延迟为 254 毫秒；而 Faiss 的吞吐量饱和于 310 每秒查询数，PipeANN-Filter 则为 58 每秒查询数。KuroANN 处理该数据流的速度是 Faiss 的 1。6× 和 8。6× 倍。然而，在每秒查询数低于 200 时，Faiss 的 p99 延迟与 KuroANN 相近。

5.4 深入分析

领先优势的来源。 图 16 将各频带运行产生的混合流组合起来，并将其相对于 Faiss 的领先优势分成两部分。该组合还使每种策略都能使用根据数据流选定的按数量表（阴影部分）。第一部分是在内存中的工作：在每个目标下，KuroANN 的引擎都比 Faiss 在相同按数量路由下的引擎更快，在 0.9 处的优势最大。这是因为它只处理探测分区中的匹配项。第二部分是按数量表所确定的读取操作。图 16 (a) 显示，该计划的运行速度比按数量路由快 1。17× 到 1。42×倍，因为每条路径都会为读取需求最多的频带过度配置一个预算。请注意，按数量表会改变基线的设置，但不会改变每种设置下的工作量。因此，当它们使用一个按数量表时，该计划仍然比 Faiss 和 PipeANN-Filter 快超过 2× 。图 16 (b) 显示，这两个部分会相乘，因为在该计划下，倒排列表获得的收益大于按数量路由下的收益。

扫描机制的影响。 图 17 在扫描开销较大的固定设置下，逐一加入扫描机制。使用大页的前瞻预取使吞吐量提高约 5× ，在 0.2% 和 1% 时尤其如此，因为不使用预取时，扫描几乎会在每个候选项上等待内存。在 17.5% 时，按位置排序的倒排列表使吞吐量提高 5。31× ，这是因为此时扫描只会访问探测分区中匹配的编码。在 0.01% 时，由于查询扫描的编码很少，这两个步骤总共只带来 5% 的提升。

以成本模型取代实测数据行。 数据库优化器

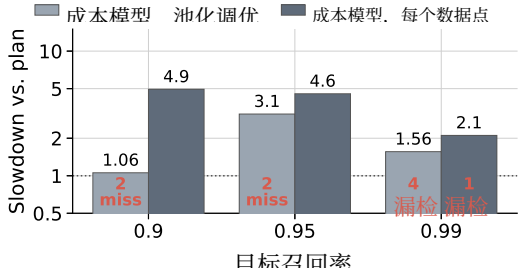
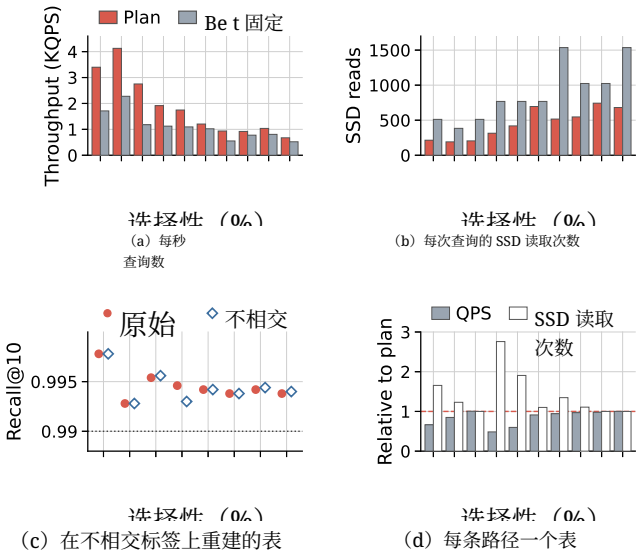


图18：成本模型计划。柱状图中的数字表示低于目标值的10个区间中所包含的区间数量。



（a）每秒查询数

（b）每次查询的 SSD 读取次数

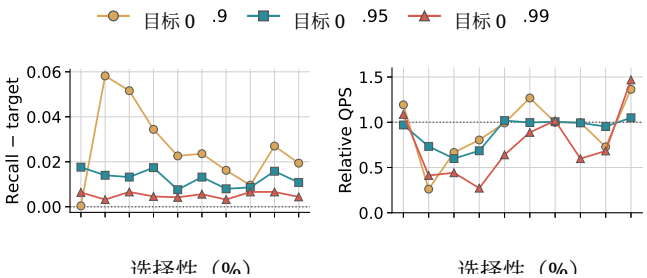
（c）在不相交标签上重建的表

（d）每条路径一个表

图 19：校准停止的影响。按计数停止使计划速度快于最佳固定预算，而停止表可以迁移到其他标签，但不能迁移到其他计数。

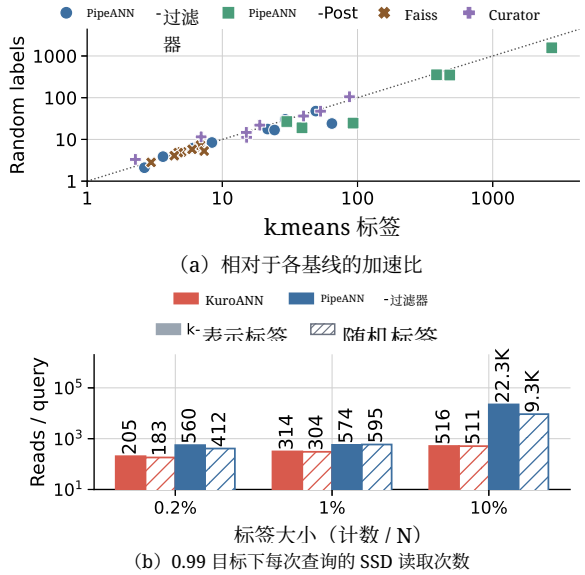
会从成本模型推导计划，因此我们也在 KuroANN 的引擎上构建了这类计划。每个计划都会选择扫描更少编码的路径，探测预计包含固定数量匹配项的最少分区，并沿每条路径读取一个池而不会提前停止。图 18 将它们与图 16的数据流上的计划进行了比较。在针对校准点汇总后的召回率调优常数后，它们在每个目标以下少运行两个到四个区间，速度为原来的 1。06×至 3。13×倍。针对每个点进行调优后，它们比该计划少漏检的区间数不超过一个，但速度为原来的 2。11×至 4。94×倍。这是因为，探测采用一条规则且每条路径只有一个池，无法跟随预算的召回率随计数变化的方式。

校准停止的影响。 图 19 （a、b）将 0.99 计划与同一引擎上每个区间的最佳固定预算进行了比较，即 AnalyticDB-V 的策略 [56]在使用精确计数时，计划快 1。18× 到 2。33× 倍，因为大多数查询会提前停止，并读取少 1。10× 到 2。97× 个向量。因此，收益来自每个查询的计数，而不是调优，因为固定预算对每个区间而言都是最佳选择。图 19 （c）在同一引擎的不相交标签上重建每个停止表



（a）召回率高于目标　（b）相对于冻结计划的每秒查询数

图 20：校准未覆盖的区间。这些计划仍能满足每个目标，但吞吐量大多低于冻结计划。



（b）0.99 目标下每次查询的 SSD 读取次数

图 21：独立于向量的标签。KuroANN 在随机标签上同样领先于每个基线，其读取次数取决于计数，而不是标签的绘制方式。

计数；八个区间都具有该计数，召回率变化最多为 0.0016。这是因为表会校准 PQ 编码的估计误差，而该误差几乎不取决于选择这些编码的标签。不过，表确实取决于计数。图 19（d）让路径的每一行都使用单一计数对应的表。这样，计划最多会多读取 2。8× 个向量，运行速度最多慢 2。06× 倍，而召回率最多提高 0.0016。这是因为在给定池排序位置上，估计值的误差大小会随计数而变化。

校准未覆盖的区间。图 20为每个区间提供一个未使用该区间进行校准的计划，其计数随后落在其余校准点之间，或超出最后一个校准点。这些计划在每个区间都能满足所有目标。然而，冻结计划最多快 3。83×倍。这是因为，一个同时经过相邻两个校准点验证的固定设置，可以在无需停止规则的情况下服务于两点之间的计数。因此，部署时需要在常见计数附近设置校准点。

标签与向量无关。由于 k-means 标签可能偏向某个分区索引，我们也按照 0.2%、1% 和 10% 标签的大小，均匀随机生成标签的独立版本——。冻结计划在两种标签上都达到了所有目标，图 21 (a) 显示，KuroANN 在两种情况下都优于所有基线。图 21 (b) 显示，在计数相同的情况下，KuroANN 对两种标签读取的向量数量都大致相同，而 PipeANN-Filter 最多会多读取 2 倍。4×。这是因为分区索引的工作量取决于计数，而图的工作量则取决于匹配项之间的连接方式。

6 相关工作

SSD 上的向量搜索。 大多数基于 SSD 的系统会搜索与原始向量一起存储的图 [10,11,46,54]。SSD 上的分区索引会读取整个分区 [3,17,61] 或对压缩候选项进行重排序 [48,60,64,65], 以上方法都不使用过滤器。

过滤向量搜索。在 SSD 上，图系统会对结果进行后过滤，或将图加密集 [12],将标签整合进图中 [9],或为匹配项添加平面扫描 [50]。近期的一个系统 NemoANN [26], 会在内存中保留每个邻居列表的截断副本，但其内存开销很大。他们的工作依赖匹配项之间的路径。在内存中，已过滤图会将谓词无关图加密集 [37] 或添加过滤感知边和子索引 [30,32,34,53,57,59,73]。VBASE [66] 和 E2E [58] 会通过宽松单调性检查或预测的距离计算预算来停止已过滤遍历。Curator [22] 会在聚类树中为每个标签的匹配项保留有序列表，我们的倒排列表将其应用于平面分区。IVF^2 [25], UBASE [49],和向量引擎 [39, 52, 55] 会根据计数或成本估计值选择路径。同样，Pinecone [18], AnalyticDB-V [56], 和 Lu 等人 [33] 会将搜索预算与选择性关联起来，正如数据库优化器会根据选择性选择计划一样 [19,51]。

每次查询的搜索开销。Quake [36] 根据几何模型为每次查询设置探测数量，并且仅初步探讨了如何根据每个分区的匹配项数量进行扩缩放。Helmsman [17] 使用学习模型预测探测数量，而 Ada-ef [63] 则从离线表中查找图的搜索宽度。此外，LAET [28],、DARTH [1],和 Auncel [67] 会在模型预测达到目标召回率的位置停止。多步搜索 [44], 、RaBitQ [8,60],、TRIM [45],和 DADE [6] 则根据距离界或采样误差分位数停止搜索或跳过重排序。它们都无需过滤器即可运行。相比之下，KuroANN 根据每次查询的计数和目标召回率来设置读取次数，包括何时停止。

7 结论

我们介绍了 KuroANN，这是一款面向固态硬盘的过滤向量搜索引擎，能够在高召回率下避免不必要的 SSD 读取。KuroANN 的核心思想是，根据首次读取前已知的匹配数量，设定每个查询的搜索范围以及停止读取的时机。其内存操作只涉及匹配项。我们希望这项工作能为研究社区带来启发，揭示过滤器——通常被视为障碍——能够指示查询需要读取多少内容。

致谢

本研究由韩国国家研究基金会（NRF）资助，资助经费来自科学技术与信息通信部（编号：RS-2025-00522990）。Gyuyeong Kim 为通讯作者。

参考文献

[1] Manos Chatzakis、Yannis Papakonstantinou 和 Themis Palpanas。DARTH：通过提前终止实现近似最近邻搜索的声明式召回。*ACM* 数据管理会议录，3(4)，2025年。第242篇文章。

[2] Jianlyu 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、多功能、多粒度文本嵌入。载于计算语言学协会论文集：*ACL 2024*，第2318–2335页，2024年。[3] 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。载于神经信息处理系统进展（*NeurIPS*），第34卷，第5199–5212页，2021年。[4] Shimin Chen、Anastassia Ailamaki、Phillip B. Gibbons 和 Todd C. Mowry。通过预取提升哈希连接性能。*ACM*数据库系统汇刊，32(3)，2007年。第17篇文章。[5] Paul Covington、Jay Adams 和 Emre Sargin。用于YouTube推荐的深度神经网络。载于第*10*届*ACM*推荐系统大会（*RecSys*）论文集，第191–198页，2016年。[6] Liwei Deng、Penghao Chen、Ximu Zeng、Tianfu Wang、Yan Zhao 和 Kai Zheng。用于高维近似最近邻搜索的高效数据感知距离比较操作。*VLDB* 基金会会议论文集，18(3)：812–821，2024年。[7] Matthijs Douze、Alexandr Guzhva、Chengqi Deng、Jeff Johnson、Gergely Szilvasy、Pierre-Emmanuel Mazaré、Maria Lomeli、Lucas Hosseini 和 Hervé Jégou。Faiss库。电气电子工程师学会大数据汇刊，12(2)：346–361，2026。

[8] Jianyang Gao 和 Cheng Long。RaBitQ：使用理论误差界量化高维向量，以进行近似最近邻搜索。*ACM* 数据管理会议录，2(3)，2024年。文章 167。

[9] Siddharth Gollapudi、Neel Karia、Varun Sivashankar、Ravishankar Krishnaswamy、Nikit Begwani、Swapnil Raz、Yiyong Lin、Yin Zhang、Neelam Mahapatro、Premkumar Srinivasan、Amit Singh 和 Harsha Vard- han Simhadri。Filtered-DiskANN：使用过滤器进行近似最近邻搜索的图算法。载于 *ACM万维网大会（WWW）*会议录，第 3406–3416 页，2023年。

[10] Hao Guo 和 Youyou Lu。通过将最佳优先搜索算法与固态硬盘对齐，实现低延迟图基向量搜索。发表于第 *19* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI*），第 171–186 页，2025年。

[11] Hao Guo 和 Youyou Lu。《OdinANN：在十亿规模的基于图的向量搜索中实现直接插入，以获得始终稳定的性能》。发表于第 *24* 届 *USENIX* 文件与存储技术会议（*FAST*），第 133–147 页，2026年。[12] Hao Guo、Jiwu Shu 和 Youyou Lu。《PipeANN-Filter：一种基于固态硬盘的高效过滤向量搜索系统》。*arXiv* 预印本 *arXiv:2605.17992*，2026年。[13] Jun He、Sudarsun Kannan、Andrea C. Arpaci-Dusseau 和 Remzi H. Arpaci-Dusseau。《固态硬盘未明文规定的契约》。发表于第 *12* 届欧洲计算机系统会议（*EuroSys*）论文集，第 127–144 页，2017年。[14] Yupeng Hou、Jiacheng Li、Zhankui He、An Yan、Xiusi Chen 和 Julian McAuley。《弥合语言与物品之间的鸿沟，实现检索与推荐》。*arXiv* 预印本 *arXiv:2403.03952*，2024年。[15] Zhengding Hu、Vibha Murthy、Zaifeng Pan、Wanlu Li、Xiaoyi Fang、Yufei Ding 和 Yuke Wang。《HedraRAG：针对异构 RAG 工作流协同优化生成与检索》。发表于第 *31* 届操作系统原理研讨会（*SOSP*）论文集，第 623–638 页，2025年。[16] Jui-Ting Huang、Ashish Sharma、Shuying Sun、Li Xia、David Zhang、Philip Pronin、Janani Padmanabhan、Giuseppe Ottaviano 和 Linjun Yang。《Facebook 搜索中基于嵌入的检索》。发表于第 *26* 届 *ACM SIGKDD* 国际知识发现与数据挖掘会议（*KDD*）论文集，第 2553–2561 页，2020年。[17] Yuchen Huang、Baiteng Ma、Yiping Sun、Yang Shi、Xiao Chen、Xiaocheng Zhong、Zhiyong Wang、Yao Hu、Erci Xu 和 Chuliang Weng。《聚类卷土重来：借助 helmsman 构建经济高效、高性能的大规模 ANNS》。发表于第 *20* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI*），第 1623–1639 页，2026。

[18] Amir Ingber 和 Edo Liberty。《在 Pinecone 的无服务器向量数据库中实现准确、高效的元数据过滤》。发表于*ICML* 第一届向量数据库研讨会，2025年。https://openreview.net/forum?id=UXq4z6GGYP。

[19] Yannis E. Ioannidis、Raymond T. Ng、Kyuseok Shim 和 Timos K. Sellis。参数化查询优化。载于第 *18* 届超大型数据库国际会议论文集，第 103–114 页，1992年。

[20] Hervé Jégou、Matthijs Douze 和 Cordelia Schmid。用于最近邻搜索的乘积量化。电气电子工程师学会模式分析与机器智能汇刊，33(1)：117–128，2011年。

[21] Hervé Jégou、Romain Tavenard、Matthijs Douze 和 Laurent Amsaleg。在十亿个向量中进行搜索：使用源编码进行重排序。载于 *IEEE* 声学、语音与信号处理国际会议（*ICASSP*），第 861–864 页，2011年。[22] Yicheng Jin、Yongji Wu、Wenjun Hu、Bruce M. Maggs、Jun Yang、Xiao Zhang 和 Danyang Zhuo。Curator：使用低选择性过滤器进行高效向量搜索。*ACM* 数据管理期刊论文集，4(1)，2026。文章 21。[23] Kanchan Joshi、Anuj Gupta、Javier González、Ankit Kumar、Krishna Kanth Reddy、Arun George、Simon Lund 和 Jens Axboe。I/O 直通：在 Linux 中上游化灵活高效的 I/O 路径。载于 第 *22* 届 *USENIX* 文件与存储技术会议（*FAST*），第 107–121 页，2024年。[24] Youngjin Kwon、Hangchen Yu、Simon Peter、Christopher J. Rossbach 和 Emmett Witchel。借助 Ingens 实现协调且高效的大页管理。载于第 *12* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI*），第 705–721 页，2016年。[25] Ben Landrum、Magdalen Dobson Manohar、Mazin Karjikar 和 Laxman Dhulipala。IVF^2 索引：融合经典倒排索引与空间倒排索引，实现快速过滤后的近似最近邻搜索。载于*ICML* 第一届向量数据库研讨会，2025年。https://openreview.net/forum?id=kXw8E3xT7O。[26] Nakyung Lee、Soobin Cho、Jiwoong Park 和 Gyuyeong Kim。NemoANN：借助图隧穿在固态硬盘上实现 I/O 高效的过滤向量搜索。载于 *ACM* 云计算研讨会（*SoCC*），2026。[27] Patrick Lewis、Ethan Perez、Aleksandra Piktus、Fabio Petroni、Vladimir Karpukhin、Naman Goyal、Heinrich Küttler、Mike Lewis、Wen-tau Yih、Tim Rocktäschel、Sebastian 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 神经信息处理系统进展（*NeurIPS*），第 33 卷，第 9459–9474 页，2020年。

[28] Conglong Li、Minjia Zhang、David G. Andersen 和 Yuxiong He。通过学习式自适应提前终止改进近似最近邻搜索。载于*2020*年 *ACM SIGMOD* 数据管理国际会议论文集，第 2539–2554 页，2020年。

[29] Mocheng Li、Xiao Yan、Baotong Lu、Yue Zhang、James Cheng 和 Chenhao Ma。近似最近邻搜索中的属性过滤：深入的实验研究。*ACM* 数据管理会议录，3(6)，2025年。文章 298。

[30] Zhaoheng Li、Silu Huang、Wei Ding、Yongjoo Park 和 Jianjun Chen。SIEVE：通过索引集合实现高效的过滤向量搜索。*VLDB* 基金会会议论文集，18(11):4723–4736，2025年。[31] Zhiyue Li、Guangyan Zhang、Ruochun Jin、Bojun Li、Xiaoguang Ren 和 Wenjing Yang。PathSeer：用于高效已过滤近似最近邻搜索的自适应近邻处理。*ACM* 数据管理会议录，4(3)，2026。文章 221。[32] Anqi Liang、Pengcheng Zhang、Bin Yao、Zhongpu Chen、Yitong Song 和 Guangxu Cheng。UNIFY：用于范围过滤近似最近邻搜索的统一索引。*VLDB* 基金会会议论文集，18(4):1118–1130，2024年。[33] Duo Lu、Helena Caminal、Manos Chatzakis、Yannis Papakonstantinou、Yannis Chronis、Vaibhav Jain 和 Fatma Özcan。在 PostgreSQL 数据库系统中对过滤器无关向量搜索进行深入研究。*ACM* 数据管理会议录，4(3)，2026。文章 134。[34] Jiarui Luo、Miao Qiao、Chaoji Zuo 和 Dong Deng。标签过滤近似最近邻搜索。载于 *IEEE* 数据工程国际会议（*ICDE*）论文集，第 3642–3654 页，2025年。[35] 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。电气电子工程师学会模式分析与机器智能汇刊，42(4):824–836，2020年。

[36] Jason Mohoney、Devesh Sarda、Mengze Tang、Shihabur Rahman Chowdhury、Anil Pacaci、Ihab F. Ilyas、Theodoros Rekatsinas 和 Shivaram Venkataraman。Quake：面向向量搜索的自适应索引。载于第 *19* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 25*），第 153–169 页，2025年。

[37] Liana Patel、Peter Kraft、Carlos Guestrin 和 Matei Zaharia。ACORN：针对向量嵌入和结构化数据的高性能、谓词无关搜索。*ACM* 数据管理论文集，2(3)，2024年。文章编号 120。

[38] Jeffrey Pound、Floris Chabert、Arjun Bhushan、Ankur Goswami、Anil Pacaci 和 Shihabur Rahman Chowdhury。MicroNN：一种设备端磁盘驻留可更新向量数据库。载于 *2025* 年数据管理国际会议（*SIGMOD-Companion*）伴随会议，第 608–621 页，2025年。

[39] Qdrant。查询规划，Qdrant 文档。https://qdrant.tech/documentation/search/search/#query-planning。访问日期：2026年11月2日。

[40] Qdrant。PubMed-MultiVector。https://huggingface.co/datasets/Qdrant/PubMed-MV，2026。[41] Derrick Quinn、Mohammad Nouri、Neel Patel、John Sal- ihu、Alireza Salemi、Sukhan Lee、Hamed Zamani 和 Mohammad Alian。加速检索增强生成。载于第 *30* 届 *ACM* 国际编程语言与操作系统架构支持会议（*ASPLOS*）论文集，第 15–32 页，2025年。[42] Bianca Schroeder、Adam Wierman 和 Mor Harchol- Balter。开放与封闭：一个警示故事。载于第三届 *USENIX* 网络系统设计与实现研讨会（*NSDI*），第 239–252 页，2006 年。[43] Christoph Schuhmann、Romain Beaumont、Richard Vencu、Cade Gordon、Ross Wightman、Mehdi Cherti、Theo Coombes、Aarush Katta、Clayton Mullis、Mitchell Wortsman、Patrick Schramowski、Srivatsa Kundurthy、Katherine Crowson、Ludwig Schmidt、Robert Kaczmar- czyk 和 Jenia Jitsev。LAION-5B：用于训练新一代图文模型的开放式大规模数据集。载于神经信息处理系统进展（*NeurIPS*），数据集与基准赛道，第 35 卷，第 25278–25294 页，2022 年。[44] Thomas Seidl 和 Hans-Peter Kriegel。最优多步 k 近邻搜索。载于*1998* 年 *ACM SIGMOD* 国际数据管理会议论文集，第 154–165 页，1998 年。[45] Yitong Song、Pengcheng Zhang、Chao Gao、Bin Yao、Kai Wang、Zongyuan Wu 和 Lin Qu。TRIM：通过增强的基于三角不等式的剪枝加速高维向量相似性搜索。*ACM* 数据管理期刊论文集，3(6)，2025年。文章 373。

[46] Suhas Jayaram Subramanya、Devvrit、Rohan Kadekodi、Ravishankar Krishnaswamy 和 Harsha Vardhan Simhadri。DiskANN：在单个节点上快速、准确地进行十亿点近邻搜索。载于神经信息处理系统进展（*NeurIPS*），第 32 卷，2019 年。

[47] Bing Tian、Haikun Liu、Zhuohui Duan、Xiaofei Liao、Hai Jin 和 Yu Zhang。使用 SmartSSD 实现可扩展的十亿点近似最近邻搜索。载于 *2024 USENIX* 年度技术会议（*USENIX ATC*），第 1135–1150 页，2024年。

[48] Bing Tian、Haikun Liu、Yuhang Tang、Shihai Xiao、Zhuo- hui Duan、Xiaofei Liao、Hai Jin、Xuecang Zhang、Jun- hua 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。发表于 第*23*届 *USENIX* 文件与存储技术会议（*FAST 25*），2025年。[49] Yao Tian、Yuncheng Lu、Liyao Xiong、Yuming Xu、Hao Zhang、Weichen Zhao、Xi Zhao、Bo Kuang、Dongyu Wang、Jiehui Li、Yakun Li 和 Lei Zhang。UBASE：字节跳动面向万亿级向量数据管理的人工智能搜索引擎。发表于 *ACM SIGOPS* 年度技术会议（*ATC*），2026。[50] Nitish Upreti、Harsha Vardhan Simhadri、Hari Sudan Sundar、Krishnan Sundaram、Samer Boshra 等。使用 Azure Cosmos DB 实现高性价比、低延迟的向量搜索。*VLDB* 基金会会议论文集，18(12)：5166–5183，2025年。[51] Kapil Vaidya、Anshuman Dutt、Vivek Narasayya 和 Surajit Chaudhuri。利用查询日志和机器学习进行参数化查询优化。*VLDB* 基金会会议论文集，15(3)：401–413，2021年。[52] Jianguo Wang、Xiaomeng Yi、Rentong Guo、Hai Jin、Peng Xu、Shengjun Li、Xiangyu Wang、Xiangzhou Guo、Chengming Li、Xiaohai Xu 等。Milvus：一种专用向量数据管理系统。发表于 *2021*年国际数据管理会议（*SIGMOD*），第2614–2627页，2021年。[53] Mengzhao Wang、Lingwei Lv、Xiaoliang Xu、Yuxiang Wang、Qiang Yue 和 Jiongkang Ni。一种高效且稳健的属性约束近似最近邻搜索框架。发表于 神经信息处理系统进展（*NeurIPS*），第36卷，第15738–15751页，2023年。[54] Mengzhao Wang、Weizhi Xu、Xiaomeng Yi、Songlin Wu、Zhangyang Peng、Xiangyu Ke、Yunjun Gao、Xiaoliang Xu、Rentong Guo 和 Charles Xie。Starling：一种面向数据段上高维向量相似性搜索的高输入/输出效率磁盘驻留图索引框架。

ACM 数据管理会议录，2(1)，2024年。第14篇文章。

[55] Weaviate。过滤器，Weaviate 文档。https://docs.weaviate.io/weaviate/concepts/filtering。访问日期：2026年11月2日。

[56] Chuangxian Wei、Bin Wu、Sheng Wang、Renjie Lou、Chaoqun Zhan、Feifei Li 和 Yuanzhe Cai。AnalyticDB-V：面向结构化和非结构化数据查询融合的混合分析引擎。*VLDB* 基金会会议论文集，13(12)：3152–3165，2020年。

[57] Jingyi Xi、Chenghao Mo、Ben Karsin、Artem Chirkin、Mingqin Li 和 Minjia Zhang。VecFlow：一种面向 GPU 上已过滤搜索的高性能向量数据管理系统。*ACM* 数据管理会议录，3(4)，2025年。第271篇文章。

[58] Wenxuan Xia、Mingyu Yang、Wentao Li 和 Wei Wang。E2E：通过自适应终止实现高效的已过滤近似 k 近邻搜索。载于第*32*届 *ACM SIGKDD* 知识发现与数据挖掘会议 *V.2 (KDD)*，第5602–5613页，2026年。[59] Yuexuan Xu、Jianyang Gao、Yutong Gou、Cheng Long 和 Christian S. Jensen。iRangeGraph：改进用于范围过滤近邻搜索的范围专用图。*ACM* 数据管理会议录，2(6)，2024年。第239篇文章。[60] Yuexuan Xu、Jianyang Gao、Michael Norris、Alibek Zhakubayev、Pankaj Singh、Junjie Qi、Matthijs Douze 和 Cheng Long。RaBitQ-SSD：用于 SSD 驻留式向量搜索的拆分编码与流水线 I/O。*arXiv* 预印本 *arXiv:2610.02652*，2026年。[61] 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：十亿级向量搜索的增量式原地更新。载于第*29*届操作系统原理研讨会 *(SOSP)*，第545–561页，2023年。[62] Peiqi Yin、Xiao Yan、Shiyuan Deng、Hui Li、Yifan Zhu、Xiangyu Zhi、Jingqi Mao、Ran Xu、Wenliang Zhang 和 James Cheng。DistVS：通过计算与内存解耦实现大规模向量搜索。载于第*23*届 *USENIX* 网络系统设计与实现研讨会 *(NSDI)*，第449–467页，2026年。[63] Chao Zhang 和 Renée J. Miller。面向自适应 HNSW 搜索的分布感知探索。*ACM* 数据管理会议录，2026年。

[64] Minjia Zhang 和 Yuxiong He。Zoom：基于固态硬盘的向量搜索，以优化准确率、延迟和内存。*arXiv* 预印本 *arXiv:1809.04067*，2018 年。

[65] Minjia Zhang 和 Yuxiong He。GRIP：面向向量搜索引擎的多存储、高容量优化高性能最近邻搜索。载于第 *28* 届 *ACM* 国际信息与知识管理会议（*CIKM*）论文集，第 1673–1682 页，2019 年。

[66] 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：通过松弛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*），第 377–395 页，2023年。

[67] Zili Zhang、Chao Jin、Linpeng Tang、Xuanzhe Liu 和 Xin Jin。在超大规模非结构化数据集上进行快速近似向量查询。载于第 *20* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 23*），第 995–1011 页，2023年。[68] Zili Zhang、Fangyue Liu、Gang Huang、Xuanzhe Liu 和 Xin Jin。借助重排序流水线，在超出GPU内存容量的大型数据集上快速处理向量查询。载于第 *21* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI*），第 23–40 页，2024年。[69] Haoru Zhao、Jingkai He、Jingyao Zeng、Mingkai Dong 和 Dong Du。分离图依赖关系，实现高效的十亿规模GPU向量搜索。载于第 *20* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI*），第 1563–1584 页，2026。[70] Da Zheng、Disa Mhembere、Randal Burns、Joshua Vogel- stein、Carey E. Priebe 和 Alexander S. Szalay。FlashGraph：在商用SSD阵列上处理十亿节点图。载于第 *13* 届 *USENIX* 文件与存储技术会议（*FAST*），第 45–58 页，2015年。[71] Yuhong Zhong、Haoyu Li、Yu Jian Wu、Ioannis Zarkadas、Jeffrey Tao、Evan Mesterhazy、Michael Makris、Junfeng Yang、Amy Tai、Ryan Stutsman 和 Asaf Cidon。XRP：使用 eBPF 实现的内核内存储函数。载于第 *16* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI*），第 375–393 页，2022年。[72] Zilliz。VectorDBBench（VDBBench）：用于向量数据库的基准测试工具。https://github.com/zilliztech/VectorDBBench，2023年。[73] Chaoji Zuo、Miao Qiao、Wenchao Zhou、Feifei Li 和 Dong Deng。SeRF：用于范围过滤近似最近邻搜索的分段图。*ACM* 数据管理会议录，2(1)，2024年。文章 69。