arXiv:2601.03229v1 [cs.DB] 6 Jan 2026

SpANNS：利用近存处理优化稀疏向量的近似最近邻搜索

Tianqi Zhang、Flavio Ponzina、Tajana Rosing 加利福尼亚大学圣迭戈分校{tiz014、fponzina、tajana}@ucsd.edu

摘要—**近似最近邻搜索（ANNS）是向量数据库中的一项基础操作，可在高维空间中实现高效的相似性搜索。稠密近似最近邻搜索已通过专用硬件加速器得到优化，而稀疏近似最近邻搜索仍受限于基于 CPU 的实现，阻碍了可扩展性。随着结合稀疏和稠密嵌入的混合检索系统成为信息检索（IR）流水线的标准，这一局限性日益突出。我们提出 SpANNS，这是一种用于稀疏近似最近邻搜索的近存处理架构。SpANNS 将混合倒排索引与高效的查询管理和运行时优化相结合。该架构基于 CXL Type-2 近存平台构建，由专用控制器负责查询解析和簇过滤，而支持计算的 DIMM 则在靠近数据的位置执行索引遍历和距离计算。与最先进的 CPU 基线相比，该架构可实现 15.2× 至 21.6× 倍的执行速度提升，为稀疏向量搜索提供可扩展且高效的解决方案。**

索引词—**近存处理、近似最近邻搜索、信息检索、稀疏向量。**

I. 引言

最近邻搜索（NNS）是高维数据处理中的一项基础问题，涉及识别距离给定查询最近的点。它广泛应用于多个领域，包括推荐系统、多媒体数据库和文档检索。为满足大规模数据集以及对延迟有严格要求的实际应用的需求，近似最近邻搜索（ANNS）技术应运而生，在相似性搜索的效率与准确率之间取得平衡。

近似最近邻搜索的一项关键应用是向量搜索引擎——现代信息检索（IR）流水线的核心组件。这些系统通常会为嵌入建立索引，以支持快速相似性搜索。受稠密嵌入能够捕获丰富语义内容这一优势的推动，先前的研究主要聚焦于稠密近似最近邻搜索 [1]。这促成了经过优化的稠密向量加速器和索引结构，并在生产系统中得到广泛采用。然而，一种趋势正日益显现

面向混合检索模型 [1]–[3]，这类模型结合了密集嵌入与稀疏嵌入，旨在发挥二者互补的优势，并按照图 1 所示构建处理流程。稀疏嵌入——例如 SPLADE [4],[5]，、DeepCT [6]，和 uniCOIL [7] 生成的嵌入——在可解释性、效率以及捕获罕见或域外

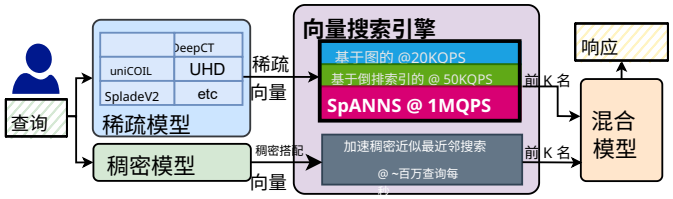


图 1：信息检索流水线向量搜索引擎中的稀疏向量近似最近邻搜索。

术语方面更有效 [8]。这些混合系统需要同时具备密集和稀疏近似最近邻搜索后端，但现有的大多数近似最近邻搜索系统和加速器仍仅针对密集场景进行了优化。

在这一流水线中，向量搜索引擎至关重要，其任务是高效搜索海量数据集。如图 1 所示，稠密向量近似最近邻搜索方法 [9]–[12]一直是研究与开发的重点，并已广泛集成到向量数据库（DB）系统 [13], [14]中。借助图形处理器 [12], [15],、现场可编程门阵列 [16],、近存储 [17], [18],及近内存技术 [19], [20], 等专用加速器，稠密向量搜索引擎的吞吐量可达每秒数百万次查询（QPS）。相比之下，稀疏向量搜索引擎仍未得到充分探索，通常依赖基于 CPU 的实现，吞吐量仅为每秒数万次查询。尽管稀疏矩阵乘法（SpMM）已针对加速器 [21],进行了高度优化，但对于稀疏向量近似最近邻搜索而言效率低下，因为它需要与整个参考集进行比较 [22]。与稠密向量近似最近邻搜索类似，倒排索引 [23]–[25] 和基于图的方法 [13], [26] 也已应用于稀疏嵌入。基于倒排索引的方法（如加权 AND（WAND） [23] 和 BWM [27], ）会选择性地跳过数据集中无关的部分；而草图方法 [28]则近似计算内积距离，以提高计算效率。

然而，这些操作严重受限于内存系统以及缺乏专用硬件支持，因此与稠密近似最近邻搜索加速器相比，吞吐量存在显著差距。如图 1 所示，稠密加速器 [19]可实现每秒数百万次查询（QPS），而稀疏图 [26]和基于倒排索引的搜索 [24], [29]引擎的吞吐量则较低。尽管稀疏嵌入具有计算优势，这一差距依然存在，

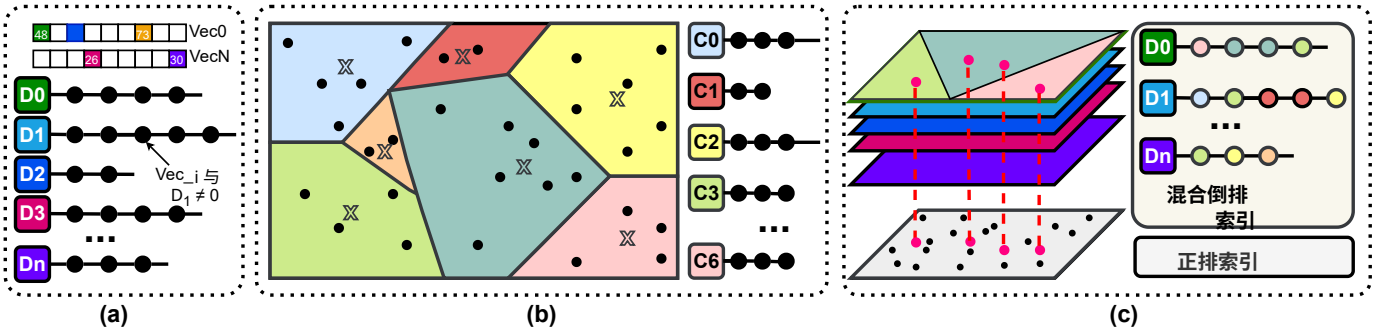


图 2：倒排索引方法示意图：(a) 基于内容，(b) 基于聚类，以及 (c) 混合设计。

尤其是在可解释性和效率至关重要的大规模应用中。

我们的工作提出了 SpANNS，这是一种硬件与软件协同设计的加速器，弥合了这一差距：

- 我们设计了 SpANNS，这是首个用于基于倒排索引的稀疏向量最近邻搜索的加速器，其速度比 Seismic CPU 基线 [24]快 15× 至 21×。据我们所知，迄今尚无其他工作为稀疏向量数据库搜索引擎开发此类加速器。
- 我们开发了一种近存处理（NMP）混合倒排索引，将非零维度与基于聚类的索引相结合，以实现高效的稀疏向量近似最近邻搜索。
- 我们设计了一种 CXL Type-2 近存处理架构，其中包含用于查询管理和簇筛选的控制器。专用双列直插式内存模块支持高效的索引处理和运行时优化，同时保留其作为普通内存的功能。

II. 相关工作：基于倒排索引的近似最近邻搜索

倒排索引最初旨在通过快速识别并排序包含相关词项的文档来加速搜索查询。形式上，对于节点集合 N 中的每个节点 n，我们定义一个特征函数或映射 $f: n \to c$，其中 $c \in C$ 是一个字符集，可将每个节点归入 C 中的且仅一个类别。于是，倒排列表 $\{I_i\}$ 定义为 $I_i = \{n | f(n) = c_i\}$，$c_i \in C$. 倒排列表有两种常见类型：内容型和基于聚类的类型。

基于内容的倒排索引根据节点是否包含特定内容来表征节点。在稀疏向量搜索中，例如 WAND [23], ，向量按照其非零维度建立索引。如图 2(a) 所示，每个稀疏向量通常仅在从 D_0 到 D_n 的维度中包含少量非零项。索引根据维度 D_i 中是否存在非零项对向量进行分组。在查询时，可以跳过与查询不存在重叠维度的向量，从而将搜索空间缩小至大约 $O\left(\frac{||q||_0}{n+1}\right)$，其中 $||q||_0$ 是查询 q 的 L_0-范数（或非零元素的数量）。然而，由于数据集中的每个稀疏向量通常包含数百个非零值，参与搜索的节点数量超过理想数量 $|N| \times \frac{||q||_0}{n+1}$ 数百倍，其中 N 是数据库中的向量数量。这些方法需要额外的内存来存储计算查询与所有相关向量之间距离的中间结果。

基于聚类的倒排索引 广泛用于稠密向量近似最近邻搜索，例如 IVF [12] 和 ScaNN [11]。这种方法通过聚类技术（如 k-means）划分空间，并将同一簇中的节点归入同一个倒排列表。如图 2(b) 所示，数据集被划分为七个簇，因此倒排列表中有七个数组。在近似最近邻搜索过程中，如果某些簇的聚类质心距离查询点过远，查询就可以跳过这些簇，从而缩小搜索空间。搜索准确率取决于簇的数量：簇的数量越多，计算质心与查询点之间距离所花费的时间越少，但假阴性聚类跳过的可能性越大。然而，对于稀疏向量而言，其维度通常显著更高（数万至数十万维），且分布更加稀疏，使用较大的簇会带来挑战，导致准确率降低或查询速度提升有限。

已有研究使用 ASIC [30], 近存计算 [31], [32] 和 FPGA [16], [33],来加速倒排索引，但这些设计针对的是稠密向量近似最近邻搜索，在稀疏场景下表现不佳。首先，它们假设进行稠密距离计算，这对于仅包含少量非零项的高维向量来说效率低下（例如，文本嵌入中的非零项通常不到 5% [4]）。其次，它们依赖基于聚类的索引，其中每条记录只属于一个列表；而在基于内容的索引中，记录会出现在多个列表中，因此需要随机访问。在这种访问模式下，其批处理优化访问模式会失效。

总之，WAND 等基于内容的倒排索引方法依赖稀疏性和高影响力维度来进行有效剪枝。然而，SPLADE 及类似模型会生成包含许多权重较小的非零维度的嵌入，这会削弱 WAND 的剪枝能力，并增加候选项评估。相比之下，基于聚类的倒排索引在高维稀疏空间中表现不佳，尤其是在数据集规模增大时，会导致效率低下。此外，使用 Intel Vtune 进行分析的结果显示，IVF 搜索高度受内存限制，其中受动态随机存取存储器限制的比例超过 80%。这些挑战凸显了对混合索引策略的需求，该策略能够结合两种索引范式的优势，同时促使我们探索近存计算技术，以缓解内存带宽限制。

III. SpANNS 概述

为解决这些问题并提升稀疏向量近似最近邻搜索的吞吐量，我们提出了 SpANNS，这是一种利用近存计算来高效利用内存带宽并挖掘固有稀疏性的软硬件协同设计。如图 2-(c) 所示，SpANNS 引入了一种适合近存计算的混合索引算法，将基于内容的索引在处理稀疏数据方面的优势与基于聚类的索引的可扩展性相结合。基于内容的索引将每个非零维度映射到包含该维度的向量集合，从而能够提前剪除无关数据。在每个倒排条目中，相似向量会进一步聚类，以支持高效查找。一个独立的正排索引存储完整的原始向量，倒排条目通过引用这些向量来确保准确检索。该索引方案背后针对近存计算友好型处理的详细算法优化将在第四节中讨论。与向量数据库类似，我们的方法优先采用内积距离，这是信息检索中使用的主要指标。

SpANNS 的架构旨在高效执行这一分层索引策略。它以 CXL Type-2 加速器的形式实现，可直接与主机系统交互。该架构采用 NMP 加速的 DIMM，并增强了 CXL 设备中的控制器逻辑。内存密集型操作被卸载到 DIMM 内的 NMP 内核，而控制器负责协调调度、DIMM 间协作以及与主机的通信。此设计最大限度地减少了主机与设备之间的数据传输，并充分利用 NMP 能力，实现高吞吐量的稀疏近似最近邻搜索。

IV. 用于稀疏向量近似最近邻搜索的混合倒排索引

稀疏向量近似最近邻搜索面临的一项关键挑战是，基于聚类的倒排索引——常用于 IVF 等稠密向量系统 [12]——直接应用于稀疏数据时会失效。这是因为传统质心会汇总一个簇中的所有非零维度，生成的稠密向量无法准确反映各条记录的稀疏结构。为解决这一问题，SpANNS 引入了一种混合倒排索引：首先采用基于内容的过滤，将搜索空间缩小到相关维度；随后在每个倒排条目中，将向量分组成簇。我们不再使用质心，而是引入轮廓，以稀疏结构和最具代表性的维度来表征每个簇。

该过程包括两个主要阶段：构建两级混合倒排索引（图 3-(a)， - ）以及基于这些预构建索[illegible]查询（图 3-(b)，1-4）。SpANNS 中的混合倒排索引建立在传统的基于内容的倒排索引（第 1 级）之上，并增加了基于聚类的倒排索引（第 2 级）。SpANNS 还保留了正排索引，以保存原始数据，正如大多数向量数据库管理系统所采用的那样 [13], [14]。

在离线索引阶段（图 3a）中， SpANN[illegible]遍历每个记录向量 x。对于每个非零维度 D_i 在 x中，x将其添加到相应的倒排列表中：$I_{D_i} = \{x \in X | x[D_i] \neq 0\}$。

接下来，检查每条记录$x \in I_{D_i}$，仅保留在维度 D_i 上具有前 K% 值的记录。此过滤步骤的灵感来源于

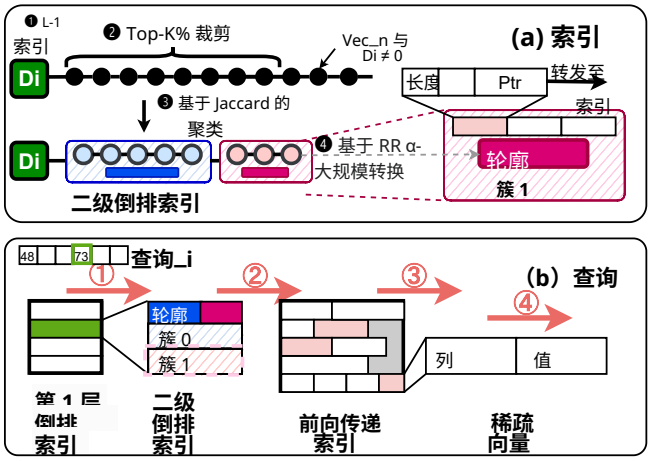


图 3：稀疏向量近似最近邻搜索的混合倒排索引

WAND [23], ，因为 $x[D_i]$ 的较小值对内积的贡献较小，通常会在查询时提前终止过程中被跳过。 然后，保留每条记录中非零值的前 $K\%$ ，以减小每个簇中非零维度并集的集合大小。为形成这些簇，SpANNS 在 k-means 中使用杰卡德距离度量，以针对稀疏性优化聚类。 对于每个簇，SpANNS 构建一个轮廓[illegible]概括稀疏向量。此过程首先对每个非零维度执行逐元素最大值运算，以生成摘要向量 m，其中$m[j] = \max_{x \in I_{D_i}} x[j]$。从 m中选择最小子集 s ，以满足以下约束：其 $L1$-范数至少保留 α 的 m’s $L1$-范数，该方法遵循先前工作中的 α-massive 摘要方法 [24]:

$$s = \arg\min_{s \subseteq m} ||s||L_0, \text{ s.t. } ||s||L_1 \geq \alpha||m||_{L_1}.$$

然而，为确保簇中每条记录都得到公平表示，SpANNS 提出了基于轮询的 α-massive 方法来构建轮廓。具体而言，我们以轮询方式在各条记录之间选取最大的非零维度，从而使轮廓与各条记录之间的非零维度保持均匀重叠。

查询期间（图 3b），SpANNS 在所提出的 NMP 加速器中执行以下步骤：1 探查第一级基于内容的倒排索引，并按查询值的大小顺序处理各维度。这反映了维度值越大，通常对内积结果的贡献越大的事实。2 对于每个维度，将查询与相关簇的轮廓进行比较；如果查询与轮廓之间的距离过大，就剪枝掉那些不太可能匹配的簇。3 对于剩余的簇，从正排索引中获取候选记录。4 最后，通过计算候选记录与查询的内积，对候选记录重新排序。

为了使数据结构适合近存计算，我们优化了布局和访问模式，以减少内存流量并提升局部性。首先，将同一维度的轮廓连续存储，从而在查询步骤 2 中实现顺序访问，这与近存计算的带宽高效

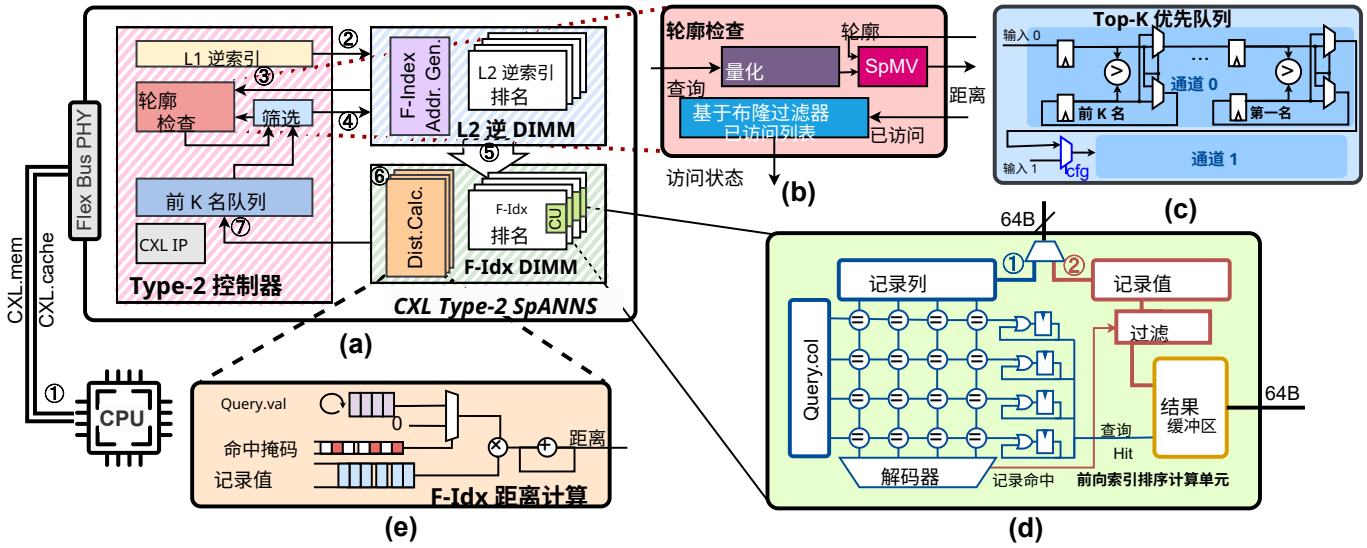


图 4：SpANNS 的架构：（a）数据流和组件；（b）Type-2 控制器中的轮廓检查逻辑；（c）可配置的 Top-K 优先队列；（d）正排索引排序中的数据压缩计算单元；（e）正排索引 DIMM 中的距离计算模块。

顺序数据访问模式。其次，我们优化了正排索引布局，以避免跨页访问开销。只要向量能够容纳在单个内存页中，就将其完整存放在该页内（在我们的设计中，内存页即 DRAM 页）。这样可以避免为单个向量读取多个页面；在近存计算环境中，内存页的粒度会显著影响访问效率，因此这点尤为重要。

V. SpANNS 近存加速器

A. 架构概述

图 4-(a) 展示了 SpANNS 的架构。SpANNS 是一款针对高效稀疏近似最近邻搜索设计的 CXL（Compute Express Link）Type-2 加速器。CXL Type-2 加速器支持高速内存内处理以及主机间的数据共享，最大限度地减少客户端 CPU 与 SpANNS 之间的数据传输，并免去复制已建立索引的数据库的需要。该架构由一个 Type-2 控制器以及与之相连的专用内存模块组成——二级倒排（L2Inv）索引 DIMM 和正排索引（F-Index）DIMM；它们协同执行两级混合索引处理。内存密集型操作分布在这些 DIMM 上，以最大限度地提高效率和吞吐量。

Type-2 控制器 通过管理一级索引访问、筛选候选项并支持 DIMM 间直接通信来协调查询过程。它使**L2 逆 DIMM** 能够将候选指针直接转发到**F-Index 双列直插式内存模块**，从而绕过 CPU 并减少数据移动。**F-Index 双列直插式内存模块**存储完整记录，并包含用于执行最终距离计算以完成最终重排序的计算单元。

B. SpANNS 数据流

图 4-(a) 展示了 SpANNS 中的数据流，该流程依次经过以下步骤 1 至 7。 1 CPU 解析查询，并按值对其非零维度进行排序。由于查询中的非零维度数量较少（通常为 10–50 个），该步骤开销较低且更适合由主机的低延迟、高频核心执行。将其卸载到加速器上几乎无法带来收益。软件管理缓冲区将解析后的查询排队，以便将预处理与近存处理执行解耦，从而实现异步流水线操作。 2 随后，排序后的查询被发送至 Type-2 控制器，由其探查一级倒排索引，以确定相关维度。 3 Type-2 控制器将查询与存储在 L2 逆 DIMM 中的簇轮廓进行比较，以确定哪些簇与查询足够接近，值得进一步处理。 4 Type-2 控制器通过比较每个簇的轮廓与查询之间的距离来筛选簇。系统维护一个前 K 名队列，用于保存最有希望的记录。如果距离超过阈值（定义为 β 乘以前 K 名队列中当前第 k 个最佳内积），则丢弃该簇。对于通过筛选的簇，已在正排索引中处理的记录会被筛除，从而确保只有唯一且相关的记录被发送至 L2 逆 DIMM。 5 L2 逆 DIMM 将所选候选记录的指针直接发送至 F-Index 双列直插式内存模块，从而减少数据移动并避免经过 CPU。 6 F-Index 获取完整记录，并利用共享的非零维度计算精细内积。 7 最终得分会加入前 K 名队列，并返回至主机。

C. SpANNS 设计细节

Type-2 控制器负责访问存储在 1 MB 缓冲区中的一级倒排索引，如图 4-(a) 所示。它最多可存储 256K 个条目，通常足以满足大多数应用的需求。例如，BERT 模型的词汇表大小为 30,522 [35]，完全在此容量范围内。如果数据集维度超过 1 MB 的限制，控制器会在页级采用 LRU 替换策略，以高效管理内存。详细的轮廓检查如图 4-(b) 所示。为评估簇相关性，首先将查询量化为 16 位定点数

格式，以减小数据量并实现高效计算。随后，稀疏矩阵-向量乘法（SpMV）单元计算量化后的查询与已存储簇轮廓之间的距离。我们按照 [36] 中所述的架构实现 Sp MV 单元。为避免重复检查，基于布隆过滤器的已访问列表使用紧凑的位数组和 [37], 中所述的轻量级整数哈希函数跟踪先前处理过的簇；这些哈希函数由异或、移位和加法操作构成，均适合硬件实现。计算出的距离会发送到 Top-K 优先队列（图 4c），该队列会随着查询的进行动态更新最佳候选项，并设置一个阈值（基于第 k 个最佳距离），用于基于轮廓的聚类筛选，以排除相关性较低的簇。Top-K 队列包含 M 个并行通道，这些通道可以独立运行，也可以合并以支持更大的 Top-nK 结果。这种灵活的设计使 SpANNS 能够适应不同的检索需求。

L2Inv DIMM 采用 Ellpack 格式，将轮廓存储在每个二级倒排索引条目的开头 [38]。轮廓矩阵通过将稀疏轮廓向量堆叠成按行排列的稠密矩阵构成，其中大多数非零值集中在少数几列中。Ellpack 格式通过对齐各行中的非零值，高效处理这种结构，从而实现紧凑存储和并行访问。这些 DIMM 还包含一个由查找表实现的地址生成器，用于转换候选指针并筛除距离较远的簇，只将相关候选项转发到 F-Idx DIMM。

F-Idx DIMM存储完整的正排索引并执行最终距离计算。在 F-Idx DIMM 中，秩级计算单元（图 4-(d)）由比较器阵列（蓝色）和压缩格式筛选器（红色）组成。1 比较器阵列将查询的非零维度索引与记录的非零维度索引进行匹配，以确定共有的非零维度。 2 筛选器阶段提取记录中匹配维度对应的值。命中掩码和筛选后的值都会暂存于结果缓冲区中，使动态随机存取存储器访问与计算流水线相互独立。随后，距离计算模块（图 4e）执行内积计算。利用比较器阵列生成的命中掩码，该操作只需对匹配的非零维度进行乘累加（MAC）。该模块在 $||q||_0$ 个周期内完成。DDR5 中的每个秩由多个裸片组成（例如，采用 $\times$16 位宽配置时使用四个裸片）。为避免裸片间通信，SpANNS 会发出多次突发读取，每次读取 4×64 字节。这种数据布局确保每条记录的列值对存储在同一裸片中，从而最大限度地降低延迟。对于 DDR5 $\times$16，每个动态随机存取存储器裸片的页大小为 2 KB，因此可以将正排索引打包到 8 KB（2 KB $\times$ 4）存储块中（见第 IV 节），从而每条记录只需执行一次行激活（ACT）命令。

D. 查询感知运行时优化

由于查询的结构（例如非零值的分布和 L0 范数）无法在索引构建时确定，

表 I：SpANNS 评估的配置参数

参数	值
DRAM 配置	8 Gb x16 DDR5-4800
时序 (tRCD-tCAS-tRP)	34-34-34
通道 / 每通道秩数	8 / 8
L1 逆索引大小	256K 个条目
L2 逆 DIMM : F-Idx DIMM 比率	1 : 3

SpANNS 提出了两种查询感知运行时优化，以实现更高效的处理。

动态正排索引距离计算：在 F-Idx DIMM 中，如第 V-C 节所述，距离计算的延迟受查询中非零值数量的影响。基于这样一个事实，还可以进一步优化：稀疏查询与记录之间的内积距离计算具有对称性，即无论遍历查询还是记录，操作都会得到相同的结果。根据 $||r||_0$ 或 $||q||_0$ 中较小者，系统会从一个向量生成命中掩码，并从另一个向量访问值，从而用更少的周期完成距离计算。这种双模式灵活性使 SpANNS 能够在运行时高效适应不同的稀疏模式。

负载均衡的正排索引检查：正排索引检查可以分布到不同的排名上并发处理，每个排名由候选指针决定。严格按照 L2Inv 索引规定的顺序执行 F-Index 检查，便可利用当前前 K 名距离，根据查询与其轮廓之间的距离来判断是否可以跳过某个簇。然而，这种严格的顺序要求在处理下一个簇之前先完整处理当前簇，从而形成可能导致处理停滞并降低 F-Idx DIMM 利用率的屏障。为解决这一问题，SpANNS 使用多个延迟队列放宽严格的处理顺序约束，从而允许无序执行 F-Idx 记录检查。这可以减少停滞并提高 F-Idx DIMM 利用率，但由于延迟处理会降低基于 top-K 距离的提前终止准确率，可能需要检查更多簇。平衡延迟队列可确保效率，同时尽量减少不必要的簇评估。相关权衡在第 VI-C 节中进行了分析。

VI. 实验结果

A. 实验设置

SpANNS 设置： 我们基于 Ramulator2 [39] 实现了一个周期精确的 SpANNS 模拟器，用于评估性能，配置详情见表 I。SpANNS 逻辑采用 SystemVerilog 开发，并使用 Design Compiler 在 ASAP 7 纳米工艺节点 [40]下评估时序和面积指标。集成到 DRAM 堆栈中的计算单元以 300 MHz 的频率完成综合，与 DDR5 内部频率相匹配。考虑到 DRAM 制造中的晶体管密度低于逻辑工艺，我们对集成在 DRAM 裸片中的 SRAM 和计算单元的面积开销均应用了 $10\times$ 缩放因子 [41]。其中的额外逻辑

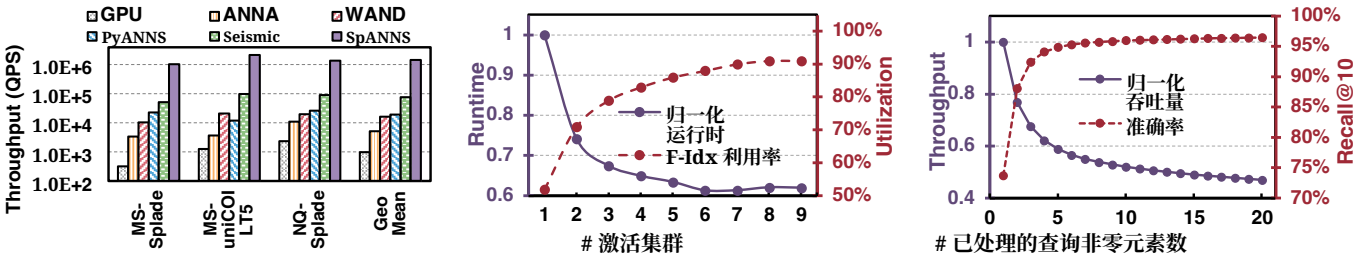


图 5：SpANNS 与图形处理器 [38], ANNA [30], WAND [29], PyANNS [26], 和 Seismic [24] 的吞吐量。图 6：负载均衡权衡：激活簇对运行时和 F-Idx DIMM 利用率的影响。图 7：已处理的非零查询元素对吞吐量和准确率的影响。

Type-2 控制器以 1 GHz 的频率完成综合。我们使用 FinCACTI [42]对基于 SRAM 的缓冲区面积进行了建模。

基线： 我们将 SpANNS 与五种基线进行了评估：(1) 图形处理器：由于缺少面向稀疏近似最近邻搜索（SANNS）的图形处理器索引结构，我们使用 cuSPARSE [38]实现了穷举搜索基线。(2) ANNA [30]：一种最初为稠密向量设计的基于聚类的倒排索引加速器。由于目前没有 SANNS 加速器，我们通过将稀疏数据集转换为稠密格式来评估它。(3) WAND [23] 算法：在 Knowhere [29]，（向量数据库 [14],的向量引擎）中优化的 SANNS 算法，在 CPU 上运行；(4) PyANNS [26], ：一种基于图的、类似分层可导航小世界的 SANNS，在 CPU 上运行；(5) Seismic [24], ：一种基于倒排索引的 SANNS，在 CPU 上运行。所有 CPU 基线均在配备 80 个线程和 512 GB DDR5 内存的 Intel Max 9460 上进行评估。图形处理器基线使用了一块 80 GB 的 NVIDIA A100。

数据集： 我们在三个预嵌入数据集 [43]上评估了 SpANNS：MS MARCO 数据集 [44] 分别使用 SPLADE [4]和 uniCOIL-T5 [7],进行嵌入，记为 MS-Splade 和 MS-uniCOILT5（880 万条记录、7000 个查询）；以及来自 BEIR [45], 并使用 SPLADE [4],进行嵌入的自然问题数据集，记为 NQ-Splade（270 万条记录、8000 个查询）。

B. SpANNS 性能

图 5 展示了 SpANNS 在三个数据集上的吞吐量，并将其与基线方法进行了比较。我们进行了网格搜索，以优化每种算法的参数，在保持召回率@10 $>$ 0.9的同时实现最佳吞吐量。与基于 CPU 的索引 Seismic [24], 、PyANNS [26], 和 WAND [29], 相比，SpANNS 的速度分别快 15–22× 倍、46–180× 倍和 69–101× 倍。与在 GPU 上进行穷举式正排索引计算相比，SpANNS 平均快 1500×倍。尽管 GPU 擅长加速稀疏矩阵计算，但对正排索引进行朴素的穷举搜索效率低下，因此 SpANNS 能够实现如此显著的加速。即使是我们实验中速度最慢的 CPU 索引（WAND [29]），其速度也最高可达到 GPU 的 32.5× 倍。就我们所知，目前尚不存在能够使用基于索引的方法加速稀疏向量近似最近邻搜索的 GPU 库，例如向量数据库搜索引擎中采用的方法。

此外，SpANNS 的性能也优于基于聚类的倒排索引加速器 ANNA [30] ，最高可达 280× 的加速比。这一性能提升源于两个关键因素：(a) SpANNS 支持优化的稀疏向量距离计算，避免了 ANNA 中密集算术运算带来的开销；(b) 与仅聚类索引相比，SpANNS 的混合索引设计显著减少了所需的距离计算次数。

当嵌入模型生成的稀疏嵌入更为稀疏时，倒排索引方法（SpANNS、WAND [23], 和 Seismic [24]）往往优于基于图的方法（PyANNS [26]）。例如，在 MS MARCO 数据集上，与 SPLADE 生成的嵌入相比，uniCOIL-T5 生成的嵌入包含的非零值少了 1.8× 。对于 MS-uniCOIL-T5, SpANNS 180×比 PyANNS [26] 快，而对于 MS-SPLADE 则快 a 46× 。

C. 负载均衡分析

严格维持 F-Index 检查的顺序（单个激活集群）会导致 F-Idx DIMM 利用率低，因为严格排序造成的屏障会使各排名等待前面的簇完成，从而停滞不前。严格执行 F-Index 检查顺序，即一次只处理一个簇，会导致 F-Idx DIMM 利用率低。这种严格排序会形成屏障，使其他排名中的处理单元在等待当前簇完成时处于空闲状态。图 6 展示了放宽这些限制对 F-Idx DIMM 利用率和运行时的影响。只有一个激活集群时，约 50% 的 F-Idx DIMM 处理单元和带宽处于空闲状态。放宽簇的严格处理顺序可以提高利用率，但代价是需要进行额外的簇评估。同时激活更多簇会降低 L2Inv 索引的过滤效率，导致对相关性较低的簇进行不必要的处理。然而，当激活簇的数量少于五个时，利用率提升带来的收益超过了额外簇检查的开销。

基于这一分析，SpANNS 在其最优配置中会同时激活五个簇。在此配置下，F-Idx DIMM 的利用率达到约 90 %，运行时显著缩短。我们观察到，多处理少量簇对准确率的影响可以忽略不计，召回率的提升不到 0.2%。

D. 查询优化的提前终止

图 7 显示，SpANNS 仅处理部分查询元素即可提前终止。SpANNS 按值降序处理查询维度，因为值较大的维度对内积的贡献更大。图 7 显示，仅处理值最高的

表 II：SpANNS 开销

组件	面积（mm^2）	功耗（W）
Type-2 控制器	0.818	1.22
L2Inv DIMM	0.117	0.015
F-Idx DIMM	0.503	0.127

五个维度后，它就达到了 95% 的 Recall@10。继续处理更多维度，吞吐量会下降近 20%，而准确率几乎没有提升。这种提前终止策略与最终重排序相结合，使 SpANNS 能够在保持高准确率的同时减少计算量。

E. 面积与功耗开销分析

表 II 展示了 SpANNS 各组件的面积和功耗。作为一款 CXL Type-2 设备，所提出的设计仅增加了 0.82 mm2 的面积和 1.22 W 的功耗，远低于配备 16 个 ARM 核心的商用 Type-2 加速器 [46], ，后者仅每个核心就占用约 2.5 mm2 [47]。DIMM 的额外功耗为每个 DIMM 0.015 W 和 0.13 W，低于每个 DIMM 典型 10 W 功耗的 2%，因此对整个系统的影响极小。在我们的实验中，所有三个数据集的 SpANNS 索引都可在 CPU 上 15 分钟内构建完成，而构建 PyANNS [26] 索引则需要 4 小时。主要原因在于，我们的设计仅对裁剪后的 L1 索引列表进行聚类，而 PyANNS 必须搜索整个数据集。

VII. 结论

SpANNS 弥补了稀疏向量搜索加速领域的空白，其性能显著优于 CPU，且可扩展性与稠密加速器相当。其混合倒排索引和近内存架构能够为向量搜索引擎高效执行稀疏近似最近邻搜索。该架构将内存密集型任务卸载到支持近内存处理的 DIMM，并通过 DIMM 间直接协调，最大限度地减少主机与设备之间的数据移动。

VIII. 致谢

本研究部分由 PRISM 和 CoCoSys 资助；它们是 JUMP 2.0 中心，该项目由 DARPA 赞助，属于 SRC 计划。本研究还获得了美国国家科学基金会的资助，项目编号为 2112665、2112167、2003279、2120019、1911095、2052809 和 2211386。

参考文献

[1] J. Guo、Y. Cai、K. Bi、Y. Fan、W. Chen、R. Zhang 和 X. Cheng，“Came：面向第一阶段检索的混合专家模型竞争式学习”，*ACM*《信息系统汇刊》，2025年。[2] Y. Yang、P. Carlson、S. He、Y. Qiao 和 T. Yang，“结合稀疏文本检索的基于聚类的部分稠密检索”，载于第*47*届国际*ACM SIGIR*信息检索研究与发展会议论文集，2024年。[3] P. Mandikal 和 R. Mooney，“稀疏与稠密相遇：一种增强科学文档检索的混合方法”，*arXiv:2401.04055*，2024年。[4] T. Formal、B. Piwowarski 和 S. Clinchant，“SPLADE：用于第一阶段排序的稀疏词汇扩展模型”，载于第*44*届国际*ACM SIGIR*信息检索研究与发展会议论文集，2021年。[5] T. Formal、C. Lassance、B. Piwowarski 和 S. Clinchant，“SPLADE v2：用于信息检索的稀疏词汇扩展模型”，*arXiv:2109.10086*，2021年。

[6] Z. Dai 和 J. Callan，“面向第一阶段段落检索的上下文感知词项加权”，载于《第43届国际ACM SIGIR信息检索研究与发展会议论文集》，2020年。[7] J. Lin 和 X. Ma，“关于 DeepImpact、COIL 以及信息检索技术概念框架的几则简要说明”，arXiv:2106 14807，2021年。。[8] K.-R. Jang、J. Kang、G. Hong、S.-H. Myaeng、J. Park、T. Yoon 和 H. Seo，“通过二值化实现高效文本检索的超高维稀疏表示”，arXiv:2104.07198，2021年。[9] S. Jayaram Subramanya、F. Devvrit、H. V. Simhadri 等，“Diskann：单节点上的快速、精确十亿规模点最近邻搜索”，《神经信息处理系统进展》，2019年。[10] Y. A. Malkov 和 D. 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》，2018年。[11] P. Sun、D. Simcha、D. Dopson、R. Guo 和 S. Kumar，“Soar：改进近似最近邻搜索的索引”，载于《神经信息处理系统》，2023年。[12] J. Johnson、M. Douze 和 H. J´egou，“使用 GPU 进行十亿规模相似性搜索”，IEEE《大数据汇刊》，2019年。[13] “pgvector”，https://github.com/pgvector/pgvector，2024年。[14] J. Wang、X. Yi、R. Guo、H. Jin、P. Xu、S. Li、X. Wang、X. Guo、C. Li、X. Xu、K. Yu、Y. Yuan、Y. Zou、J. Long、Y. Cai、Z. Li、Z. Zhang、Y. Mo、J. Gu、R. Jiang、Y. Wei 和 C. Xie，“Milvus：专用向量数据管理系统”，载于《2021年国际数据管理会议论文集》，2021年。[15] H. Ootomo、A. Naruse、C. Nolet、R. Wang、T. Feher 和 Y. Wang，“Cagra：面向 GPU 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）。IEEE，2024年。[16] J. Zhang、J. Li 和 S. Khoram，“基于 OpenCL FPGA 的高效大规模近似最近邻搜索”，载于《2018年IEEE/CVF计算机视觉与模式识别会议》，2018年。[17] W. Xu、J. Chen、P.-K. Hsu 等，“Proxima：面向基于图的近似最近邻搜索的近存储加速”，arXiv:2312.04257，2023年。[18] Y. Wang、S. Li、Q. Zheng、L. Song 等，“NDSearch：通过近数据处理加速基于图遍历的近似最近邻搜索”，载于《2024年第51届ACM/IEEE计算机体系结构年会》（ISCA）。IEEE，2024年。[19] Z. Zhu、J. Liu、G. Dai、S. Zeng、B. Li、H. Yang 和 Y. Wang，“面向十亿规模近似最近邻搜索的分层存储器内处理架构”，载于《2023年第60届ACM/IEEE设计自动化会议》（DAC），2023年。[20] J. Jang、H. Choi、H. Bae、S. Lee、M. Kwon 和 M. Jung，“CXL-ANNS：面向十亿规模近似最近邻搜索的软硬件协同内存解聚与计算”，载于《2023年USENIX年度技术会议》（ USENIX ATC 23），2023年。[21] S. Yesil、A. Heidarshenas、A. Morrison 和 J. Torrellas，“WISE：使用机器学习预测稀疏矩阵-向量乘法的性能”，载于《第28届ACM SIGPLAN并行编程原理与实践年会论文集》，2023年。[22] S. R. Agrawal、C. M. Dee 和 A. R. Lebeck，“利用加速器实现高效高维相似性搜索”，载于《第21届ACM SIGPLAN并行编程原理与实践研讨会论文集》，2016年。[23] A. Z. Broder、D. Carmel、M. Herscovici 等，“使用两级检索流程实现高效查询求值”，载于《第十二届国际信息与知识管理会议论文集》，2003年。[24] S. Bruch、F. M. Nardini、C. Rulli 和 R. Venturini，“面向学习型稀疏表示近似检索的高效倒排索引”，载于《第47届国际ACM SIGIR信息检索研究与发展会议论文集》，2024年。[25] S. Bruch、F. M. Nardini、A. Ingber 和 E. Liberty，“连接稠密与稀疏最大内积搜索”，ACM《信息系统汇刊》，2024年。[26] Z. Wang，PyANNS，2023年。[在线]。可用地址：https://github.com/veaaaab/pyanns [27] C. Dimopoulos、S. Nepomnyachiy 和 T. Suel，“优化块最大索引的 top-k 文档检索策略”，载于《第六届ACM国际网络搜索与数据挖掘会议论文集》，2013年。

[28] S. Bruch、F. M. Nardini、A. Ingber 和 E. Liberty，“面向流式稀疏向量最大内积搜索的近似算法”，ACM《信息系统汇刊》，2023年。[29] “zilliztech/knowhere”，2024年。[在线]。可用地址：https://github.com/zilliztech/knowhere [30] Y. Lee、H. Choi、S. Min、H. Lee、S. Beak、D. Jeong、J. W. Lee 和 T. J. Ham，“ANNA：面向近似最近邻搜索的专用架构”，载于《2022年IEEE高性能计算机体系结构国际研讨会》（HPCA），2022年。[31] S. Chen、A. C. Zhou、Y. Shi、Y. Li 和 X. Yao，“Memanns：利用实用存内处理硬件提升十亿规模近似最近邻搜索效率”，arXiv:2410.23805，2024年。[32] K. Chen、R. Nadig、M. Frouzakis 等，“Reis：一种采用存储内处理的高性能、高能效检索系统”，载于《第52届计算机体系结构年度国际研讨会》，2025年。[33] W. Jiang、M. Zeller、R. Waleffe、T. Hoefler 和 G. Alonso，“Chameleon：面向检索增强语言模型的异构解聚式加速器系统”，arXiv:2310.09949，2023年。[34] C. Lassance 和 S. Clinchant，“SPLADE 模型的效率研究”，载于《第45届国际ACM SIGIR信息检索研究与发展会议论文集》，2022年。[35] J. Devlin，“BERT：面向语言理解的深度双向 Transformer 预训练”，arXiv:1810.04805，2018年。[36] G. Oyarzun、D. Peyrolon、C. Alvarez 和 X. Martorell，“用于非结构化计算流体力学模拟的基于 FPGA 缓存的稀疏矩阵-向量乘法（SpMV）”，arXiv:2107.12371，2021年。[37] B. Jenkins，“整数哈希”，2025年。[在线]。可用地址：https://www.burtleburtle.net/bob/hash/integer.html/ [38] 英伟达，“cuSPARSE”，https://developer.nvidia.com/cusparse/，2024年。[39] H. Luo、Y. C. Tu、F. N. Bostancı、A. Olgun、A. G. Ya、O. Mutlu 等，“Ramulator 2.0：现代化、模块化且可扩展的动态随机存取存储器模拟器”，IEEE 计算机体系结构快报，2023年。[40] L. T. Clark、V. Vashishtha、L. Shifren、A. Gujja、S. Sinha、B. Cline、C. Ramamurthy 和 G. Yeric，“ASAP7：7 纳米 FinFET 预测工艺设计套件”，《微电子学杂志》，2016年。[41] F. Devaux，“真正的存内处理加速器”，载于《2019年IEEE Hot Chips第31届研讨会》（HCS），2019年。[42] A. Shafaei、Y. Wang、X. Lin 和 M. Pedram，“FinCACTI：采用深度缩放 FinFET 器件的缓存架构分析与建模”，载于《2014年IEEE计算机学会超大规模集成电路年度研讨会》，2014年。[43] Seismic Datasets，2024年。[在线]。可用地址：https://huggingface.co/collections/tuskanny/seismic-datasets-6610108d39c0f2299f20fc9b[44] P. Bajaj、D. Campos、N. Craswell 等，“MS MARCO：人工生成的机器阅读理解数据集”，arXiv:1611.09268，2016年。[45] N. Thakur、N. Reimers、A. R¨uckl´e、A. Srivastava 和 I. Gurevych，“BEIR：用于零样本评估信息检索模型的异构基准”，arXiv:2104.08663，2021年。[46] “Marvell 2504 近存加速器产品简介”，https://www.marvell.com/content/dam/marvell/en/public-collateral/assets/marvell-structera-a-2504-near-memory-accelerator-product-brief.pdf，2025年。[47] M. Bruce，“Arm Neoverse V2 平台：面向下一代云计算、机器学习和高性能计算工作负载的领先性能与能效”，载于《2023年IEEE Hot Chips第35届研讨会》（HCS），2023年。