arXiv:2601.08528v1 [cs.DB] 13 Jan 2026

SVFusion：面向大规模实时向量搜索的 CPU-GPU 协同处理架构

Yuchen Peng
浙江大学†
杭州高新技术产业开发区（滨江）研究院区块链与数据安全
zjupengyc@zju.edu.cn

Dingyu Yang*
浙江大学†
杭州高新技术产业开发区（滨江）研究院区块链与数据安全
yangdingyu@zju.edu.cn

Zhongle Xie
浙江大学†
杭州高新技术产业开发区（滨江）区块链与数据安全
xiezl@zju.edu.cn

Ji Sun
华为技术有限公司, Ltd
sunji11@huawei.com

Lidan Shou*
浙江大学†
杭州高新技术产业开发区（滨江）研究院区块链与数据安全
should@zju.edu.cn

Ke Chen
浙江大学†
杭州高新技术产业开发区（滨江）研究院区块链与数据安全
chenk@zju.edu.cn

Gang Chen
浙江大学†
杭州高新技术产业开发区（滨江）区块链与数据安全研究院
cg@zju.edu.cn

摘要

近似最近邻搜索（ANNS）是信息检索和推荐等现代应用的基础。随着向量数据快速增长，面向实时向量搜索的高效索引已成为基本需求。现有基于 CPU 的解决方案支持更新，但吞吐量较低；而 GPU 加速系统虽然性能出色，却面临动态更新和 GPU 内存有限等挑战，因此在兼顾准确率与速度的持续大规模向量搜索方面存在显著性能差距。本文提出 SVFusion，这是一种面向实时向量搜索的 GPU-CPU-磁盘协同框架，可将复杂的 GPU 计算与在线更新相结合。SVFusion 采用分层向量索引架构，运用 CPU-GPU 协同处理，并结合工作负载感知的向量缓存机制，最大限度地提高有限 GPU 内存的利用效率。此外，它通过 CUDA 多流优化和自适应资源管理实现实时协调，并借助并发控制确保交错执行查询和更新时的数据一致性。实验结果表明，SVFusion 显著降低了查询延迟并提高了吞吐量：与基线方法相比，平均吞吐量提高 20.9×，延迟降低 1.3× 至 50.7×，同时在各种流式工作负载下，针对大规模数据集保持了较高的召回率。

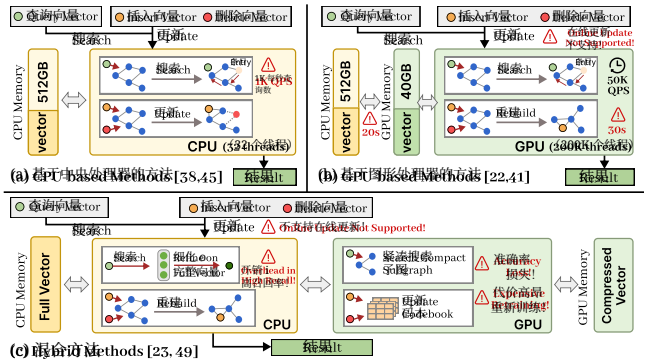


图1：现有近似最近邻搜索方法的比较。

1 引言

深度学习（DL）和大语言模型（LLM）的兴起，推动了高维向量嵌入在文本、图像和音频等非结构化数据中的广泛应用。高效检索这些嵌入向量对于个性化推荐 [29],网页搜索 [67],以及基于大语言模型的应用 [38]等任务至关重要。为了满足服务级别目标（SLO）下严格的低延迟和高准确率要求，现代向量搜索系统越来越多地采用近似最近邻搜索（ANNS） [40],，尤其是基于图的解决方案 [14, 20,32, 42, 53] ，在高维空间中兼顾准确率与计算效率。在此基础上，近期研究探索了 GPU加速优化方案，包括 SONG [69], 、GANNS [65], 和 CAGRA [47],，利用大规模并行和高内存带宽，相较于基于 CPU 的方法实现显著加速，从而支持可扩展的高吞吐量向量搜索。

与此同时，持续生成的嵌入向量呈爆炸式增长，促使向量搜索从离线建索引根本性地转向流式处理 [10, 11, 34, 39, 48]。在应急响应 [45],和检索增强生成（RAG）系统等实时应用场景中，系统持续摄取实时数据，而检索依赖于先前的结果，因此必须保持索引新鲜度，才能确保 LLM 响应连贯。京东 [39] 等电子商务平台处理数百万个独立搜索查询，这些查询必须反映库存变化。在实时视频分析和监控中，数据流每秒产生海量嵌入向量，这些向量必须持续建索引并接受查询，以执行诸如

*通讯作者：Dingyu Yang、Lidan Shou
†区块链与数据安全国家重点实验室

目标跟踪和异常检测 [51]等任务。这些应用不仅要求在持续更新期间实现实时数据同步，还要求更高的处理吞吐量和更低的延迟。

尽管高性能向量搜索取得了显著进展，但现有系统大多针对静态或缓慢演变的数据集设计，因此不适用于动态环境。如图1所示，大规模流式场景要求支持搜索、插入和删除操作 [12]。基于中央处理器的方法 [42, 53, 63] 支持实时更新，但计算开销高、内存带宽有限，随着数据集增大，可扩展性较差。相比之下，基于图形处理器的方法 [25, 47] 在静态索引上能够实现高吞吐量，但处理大规模更新时会产生显著的传输开销 [36]。此外，对于高速流式工作负载而言，完全重建索引并不切实际。

基于中央处理器的系统（便于更新但性能受限）与基于图形处理器的系统（吞吐量高但静态）之间的差异，促使人们探索统一的混合方法。近期方法 [26,57] 尝试通过将压缩向量存储在 GPU 内存中来弥合这一差距，但这会导致精度下降，无法满足高召回率要求；此外，这些方法依赖静态量化码本，处理流式更新时需要进行代价高昂的重新训练 [24]。总而言之，解决大规模流式近似最近邻搜索问题面临两项挑战：

挑战 1：GPU 内存容量限制。 现代向量数据应用处理包含数百万至数十亿个高维嵌入的大规模数据集 [6, 27], , 其规模通常超过 GPU 内存容量，导致性能不佳。例如，在 768 维度下服务 3,500 万个向量 [8] 至少需要 120GB，远超 NVIDIA A100 等高端加速器通常具备的 40GB 容量。这一限制使得数据必须频繁在 GPU 与主机内存之间移动，而 GPU 并行性带来的性能优势会因高昂的数据传输开销而大幅削弱 [55]。尽管压缩技术 [33, 57]能够降低存储需求，但通常会导致搜索准确率下降，或为维持相当的结果而增加延迟。因此，有必要构建能够有效管理数据放置的高效 CPU-GPU 协同框架。

挑战 2：频繁向量更新导致的性能下降。 大规模高频向量更新会带来两个关键的性能问题。首先，现有的基于图的方法在频繁插入和删除期间面临有限的更新吞吐量 [32, 42], , 通常依赖同步删除标记和定期图重建，这会造成延迟峰值并阻碍响应能力。在 GPU 上，由于修改图拓扑需要频繁进行内核同步 [56], , 这一问题会进一步加剧；在混合 CPU-GPU 系统中，跨设备数据传输还会带来额外开销 [56, 57]。其次，随着频繁更新使图结构逐渐恶化，搜索准确率会随时间下降，进而导致并非最优的遍历路径 [7, 47]。

针对这些挑战，一个问题随之而来：我们能否在避免数据压缩的同时，高效支持高频向量更新？在本文中，我们提出了 SVFusion，这是一种面向大规模流式向量搜索的 CPU-GPU 协同框架，将图形处理器加速与实时更新相结合，在保持低延迟的同时实现高吞吐量。SVFusion 建立在三项关键创新之上：**(1)分层向量索引结构**，在图形处理器、CPU 内存和磁盘之间组织向量数据与图拓扑，以最大化计算效率。我们设计了实时协调机制，结合 CUDA 多流优化与自适应资源管理，实现高吞吐量与低延迟。**(2) 工作负载感知向量放置机制**，自适应地确定是在 CPU 与图形处理器之间传输向量，还是在所需向量不位于 GPU 内存中时直接在 CPU 上执行计算。通过同时评估访问模式与图结构，该机制在数据局部性和传输成本之间实现了更优的权衡，在动态工作负载下提供高效率和稳定的性能。**(3) 实时更新并发控制**，支持安全且高性能的并发查询与更新。我们通过结合细粒度锁定与跨层同步提供一致性保证，并采用多版本机制确保一致性与卓越性能。

基于这些技术，SVFusion 实现了具备更新能力的高性能实时向量搜索。总而言之，我们的贡献如下：

- 我们提出了*SVFusion*，一种用于流式向量搜索的新型图形处理器-中央处理器-磁盘协同框架，它将分层索引与实时协调相结合，以最大限度地提高异构内存层级中的计算效率。（§3）。
- 我们设计了一种工作负载感知的放置机制，可高效平衡数据缓存与计算决策，从而优化动态工作负载下的性能。（§4）
- 我们提出了一种并发控制协议，通过细粒度锁定和多版本同步技术来维护数据一致性。（§5）
- 我们实现了*SVFusion*并在多种流式工作负载下进行了全面评估，以验证其有效性和效率。结果表明，与基线方法相比，我们的框架在保持更高召回率的同时，搜索吞吐量最高提升 9.5×，插入吞吐量提升 71.8×，延迟降低 1.3× 至 50.7×。（§6）

2 预备知识

我们首先定义近似最近邻搜索（ANNS）问题及其流式变体（SANNS），随后概述基于图的索引技术。表1总结了常用符号。

2.1 近似最近邻搜索

让 $X_N = \{x_1, x_2, \ldots, x_N\} \subset \mathbb{R}^D$ 表示一个由 N 个向量构成的数据集，其中每个数据点 $x_i \in \mathbb{R}^D$ 表示 D 维实值向量。任意两个向量 $u, v \in \mathbb{R}^D$ 之间的距离记为 $dist(u, v)$，采用欧几里得距离（即 L2 范数）度量。给定查询向量 $q \in \mathbb{R}^D$ 和正整数 $k (0 < k < N)$，最近邻搜索 （NNS）问题 [17] 旨在找出一个包含 k 个与 q 距离最近的向量的子集 $U_{\text{NNS}} \subset X$，并满足以下条件：对于任意 $x_i \in U_{\text{NNS}}$ 以及任意 $x_j \in X \setminus U_{\text{NNS}}$，均有 $\text{dist}(x_i, q) \leq \text{dist}(x_j, q)$。

对于大型数据集，精确最近邻搜索的计算开销过高 [60], , 因此大多数研究 [19, 20, 35, 42] 采用

表1：符号汇总。

符号	定义
X	N D维向量集合
q	一个查询向量
k	近似最近邻搜索中返回的结果数量
$G = (V, E)$	顶点集为 V、边集为 E的邻近图
X_t	时间 t时的数据集状态
X_t^G	时间 t时驻留在 GPU 内存中的向量子集
M	GPU 内存容量（向量数量）
λ_x	向量 x的未来访问次数
$F_{recent}(x, t)$	向量 x 在时间 t 的近期访问次数
$E_{in}(x)$	向量 x 的入邻居数量
α, β	预测函数中的权重参数

近似最近邻搜索 (ANNS)，以在准确率和效率之间取得平衡。给定近似因子 $\epsilon > 0$，近似最近邻搜索算法会返回一个有序集合，其中包含 k 个向量 $U_{\text{ANNS}} = \{x_1, x_2, \ldots, x_k\}$，并按其与 q 之间的距离升序排列。若 x_i^* 是 q 在 X 中的第 i 个最近邻，则对于所有 $i = 1, 2, \ldots, k$，该算法满足 $\text{dist}(x_i, q) \leq (1 + \epsilon) \cdot \text{dist}(x_i^*, q)$。

2.2 流式 ANNS

该流式近似最近邻搜索（SANNS）问题将传统近似最近邻搜索扩展到能够处理包含动态插入和删除的持续演变的数据集。SANNS 会随时间处理有序操作序列 $O = \{o_1、o_2……、o_T\}$，其中 T 表示流中的时间步或操作数量。在每个时间步 $t \in [1, T]$，向量数据集状态 X_{t-1} 上恰好执行一个操作 o_t，从而产生新状态 X_t。它支持以下四种操作类型：

- **构建**(Xinit)：$X_t = X\text{init}$，用于初始化数据集状态，其中 $X\text{init} \subset \mathbb{R}^D$，并可选择定期重建。
- **搜索**(q、k)：$X_t = X_{t-1}$。给定一个查询向量 q 和一个正整数 k，此操作从 X_{t-1} 中检索一组 k 向量，这些向量近似于 q 的真实 k 个最近邻。
- **插入**(x_t)：$X_t = X_{t-1} \cup \{x_t\}$，将新向量纳入数据集。此操作动态扩展可搜索向量空间，以容纳新数据。
- **删除**(x_t)：$X_t = X_{t-1} \setminus \{x_t\}$，移除一个现有向量。此操作通过消除过时或无关的向量来缩小搜索空间。

与真实值固定的静态数据集上的近似最近邻搜索不同，SANNS 面临着在不断演变的数据集上保持准确率，同时确保动态操作高效执行的挑战。

2.3 基于图的索引

我们定义一个有向图 $G = (V, E)$，其中 V 是顶点集合，具有 $|V| = N$，且每个顶点 $v_i \in V$ 对应数据集中的一个向量。边集 $E \subseteq V \times V$ 表示顶点之间的有向连接：$(v_i, v_j) \in E$ 表示顶点 v_i 连接到邻近顶点 v_j，边的方向从每个顶点指向其选定的邻居。对于任意顶点 $v_i \in V$，我们将其出邻居集合记为 $N_{out}(v_i) = \{v_j \in V \mid (v_i, v_j) \in E\}$，将其入邻居集合记为 $N_{in}(v_i) = \{v_j \in V \mid (v_j, v_i) \in E\}$。

基于图的近似最近邻搜索和 SANNS。基于图的近似最近邻搜索通过贪心遍历邻近图来执行查询处理。邻近图主要分为两类：K近邻图（*KNNG*），其中每个顶点恰好维护 k 条指向其近似最近邻的出边 [18], 以及相对邻域图（*RNG*），通过删除冗余边来优化连通性，同时保持度数可变 [20, 32]。给定一个查询向量 q，向量搜索从若干入口点开始，这些入口点可以随机选取 [32, 43, 47]，也可以由单独的算法提供 [23, 31]。随后，该过程会迭代探索距离 q 越来越近的邻居节点，直到满足停止条件 [32]。为了支持流式场景（SANNS），索引必须处理持续更新。在时间 t，设 $G_t = (V_t, E_t)$ 为当前图。给定插入集合 I_t 和删除集合 $D_t \subseteq V_t$，图将更新为 $G_{t+1} = (V_{t+1}, E_{t+1})$，其中 $V_{t+1} = (V_t \cup I_t) \setminus D_t$，并且 $E_{t+1} \subseteq V_{t+1} \times V_{t+1}$。

基于图的近似最近邻搜索的GPU加速。使用 GPU 可以显著加速近似最近邻搜索。在实践中，基于 GPU 的方法通常采用固定出度的 K 近邻图，以充分发挥 GPU 的大规模并行能力，因为可变出度会导致负载不均衡和计算资源利用不足 [25, 47]。关键的加速手段包括将图结构等关键元数据存储在高带宽内存中，并为每个查询分配一个线程块，而非一个线程 [25]，从而实现核心操作的大规模并行化。与基于 CPU 的方法相比，这种设计显著提升了性能。

3 概述

我们提出了 SVFusion，这是一种用于 SANNS 的 GPU–CPU–磁盘协同框架。该框架旨在解决支持实时向量搜索和更新这一根本挑战。为克服 GPU 内存容量限制，(1) 我们引入了一种**分层图向量索引**，在 GPU、CPU 和磁盘层级之间维护同步的图结构，使数据集大小增长时能够在不同内存层之间无缝迁移；(2) 我们设计了一种**工作负载感知的向量放置策略**，动态管理 GPU、CPU 和磁盘中的数据驻留与缓存，以尽量减少 PCIe 和 I/O 开销。为进一步提升高频更新场景下的性能，(3) 我们提出了一种**实时协调机制**，并采用**并发控制**，在确保各内存层级之间一致性的同时，实现高吞吐量、低延迟的并发操作。如图 2a 所示，SVFusion 通过三个关键组件实现了这些目标：

图形处理器模块通过在高带宽内存（HBM）中存储热点向量和子图，加速向量搜索与更新。我们将 CAGRA [47], ——一种最先进的、最初为静态数据集设计的基于 GPU 的索引——扩展为动态且高效支持更新的索引，并采用专用算法，支持在超出 GPU 内存容量的规模上进行实时向量搜索。

中央处理器模块通过多版本副本维护完整索引图，并将所有向量存储在动态随机存取存储器中；当中央处理器内存耗尽时，则将冷向量卸载到磁盘。该模块负责协调中央处理器计算与同步，并实现并发访问控制。

专用的**缓存管理器**会在 GPU、CPU 和磁盘层级之间管理数据放置，以优化混合工作负载下的查询性能。不同于依赖时间局部性的传统缓存策略 [46], ，我们的缓存管理器利用向量访问模式和图拓扑，动态加载和驱逐向量条目，最大限度地提高图遍历期间所访问向量的 GPU 内存驻留率。

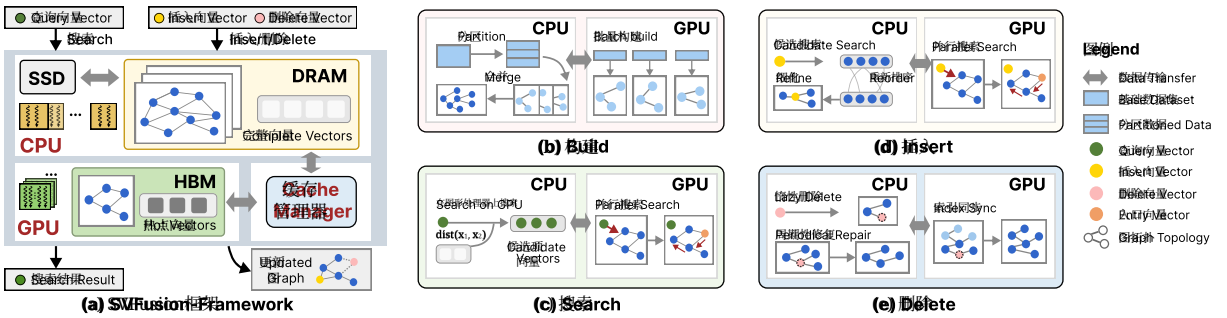


图 2：SVFusion 概览。

SANNS 工作流。图 2b-2e 展示了 SVFusion 端到端的 SANNS 工作流，该工作流通过协同的 CPU-GPU 处理完成以下四项操作：

构建（§4。2）。该索引采用受 CAGRA [47], 启发的 GPU 并行策略构建，通过执行邻居搜索和图链接，充分利用 GPU 并行性和内存带宽。对于大型数据集，该策略会对数据进行分区，在图形处理器上为每个分区构建子图，并在中央处理器上将其合并为统一索引，同时保留分区间连通性。当完整图超出 CPU 内存容量时，此操作会在有界内存窗口内合并子图，避免将整张图加载到内存中。

搜索（§4.2 & 4.3）。给定查询后，此操作会从图形处理器开始执行图遍历。**缓存管理器** 会检查所需的向量或子图是否已缓存在 GPU 内存中。如果已缓存，搜索便完全在图形处理器上进行；否则，SVFusion 会按需从中央处理器传输数据，或自适应地在中央处理器上计算距离，以减少传输开销。最后，通过排序获取 top-k 个结果并返回给用户。

插入（§5.1）。此操作会在保持图结构和搜索质量的同时整合新向量。首先，对每个向量执行基于图形处理器的搜索，以识别候选邻居；必要时按需传输数据。随后，中央处理器执行**启发式重排序** 以细化邻居选择，并通过**反向边插入** 维持双向连通性。

删除（§5.2）。此操作采用轻量级删除机制，以处理大规模向量删除请求。该机制基于两种互补策略：（1）**惰性删除**，将向量标记为已删除，而不立即更改图结构；（2）**异步修复**，通过局部拓扑感知修复和定期全局整合来恢复图连通性。

4 基于图的向量搜索与 CPU-GPU 协同处理

为处理具有动态访问模式的海量向量数据集，SVFusion 引入了分层索引架构、适应工作负载的自适应缓存机制以及实时协调策略，从而实现高吞吐量、低延迟的流式向量搜索。

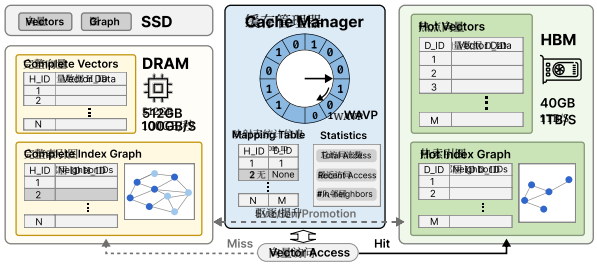


图 3：SVFusion 中的分层索引结构。

4.1 动机

分层内存管理。 由于 GPU 内存容量有限，无法完全容纳十亿规模的向量数据集。虽然压缩向量 [57, 68] 能够减少内存占用，但往往会降低搜索准确率。鉴于现实世界中的访问模式高度倾斜 [44], ，我们采用分层内存设计，将热点向量以未压缩形式缓存在图形处理器上，从而能够在海量数据集上实现可扩展的高准确率搜索。

动态缓存管理：在流式 ANNS 中，访问模式会不断变化，导致先前的热点向量变为冷向量。不同于传统缓存方法 [37, 46] 在计算前获取数据，CPU-GPU 协同处理中的向量访问决定了计算发生的位置，以及是否需要进行数据传输。因此，我们采用工作负载感知缓存机制，以便根据不断变化的工作负载优化向量放置。

4.2 基于层次图的向量索引

图 3 展示了跨越中央处理器和图形处理器内存层级的分层索引结构。现有 GPU 加速的近似最近邻搜索方法在处理大规模数据集 [47, 69], 时受限于有限的 GPU 内存容量，而我们的设计利用了中央处理器和图形处理器内存的互补特性：图形处理器内存带宽高（1 TB/s）但容量有限；CPU 内存容量大（数百 GB）但带宽较低（100 GB/s）。为弥合带宽与容量之间的差距，我们的层次化设计将向量数据分布在多个存储层级中，并通过协同处理搜索算法动态利用两种处理器，从而在大规模数据集上实现高吞吐量的近似最近邻搜索。

层次索引结构。 我们的索引采用三层 GPU–CPU–磁盘架构，可实现高效缓存、无缝数据移动，并突破图形处理器和主存的容量限制，具备更强的可扩展性。对于大小为 N 的数据集，CPU 内存维护图结构

算法 1 使用 CPU-GPU 协同处理的近似最近邻搜索

输入： 图索引 G，一批查询 $p\ Q = \{q_1, q_2, \ldots, q_p\}$，$k$ 用于前-k，以及候选池大小 $L \geq k$
输出： $K := \cup_{i=1}^{p}\{K_i\}$，其中 K_i 是 q_i 的 k 个最近邻的集合

1: **并行处理**每个查询 $q_i \in Q$
2: $C_i \leftarrow \text{InitCandidatePool}(G, L)$ ▷ 随机初始化
3: **重复**
4: $x_{curr} \leftarrow C_i$.获取最近邻(q_i)
5: $X_i \leftarrow$ 获取邻居(x_{curr}, G) ▷ 在图形处理器上
6: $X_i^{GPU}, X_i^{CPU} \leftarrow \emptyset, \emptyset$ ▷ 初始化处理集合
7: **对**每个 $x \in X_i$ **并行执行**
8: $d_id \leftarrow$ 映射(x) **或**WAVP(x)
9: ▷ 仅在缓存未命中时调用向量放置
10: $X_i^{GPU} \leftarrow X_i^{GPU} \cup \{x | d_id \neq \text{无}\}$
11: $X_i^{CPU} \leftarrow X_i^{CPU} \cup \{x | d_id = \text{无}\}$
12: $D_i^{CPU} \leftarrow$ 并行计算距离(q_i, X_i^{CPU}) ▷ 在中央处理器上
13: $D_i^{GPU} \leftarrow$ 并行计算距离(q_i, X_i^{GPU}) ▷ 在图形处理器上
14: C_i.更新($D_i^{CPU} \cup D_i^{GPU}, X_i$) ▷ 更新候选项
15: **直到** C_i 与前一次迭代保持不变
16: $K_i \leftarrow k$ 距 q_i 最近的候选项来自 C_i

$G = (V, E)$，其中包含主机端标识符（$h_id \in [1, N]$）。GPU 内存缓存了所选的一部分 $M \ll N$ 热点向量，并为其配备紧凑的设备端标识符（$d_id \in [1, M]$）。

通过将热点向量及其关联的子图保留在图形处理器上，大多数计算都可以直接由高带宽内存完成，且不会损失准确率。对于超出动态随机存取存储器容量的大规模数据集，我们的框架可以无缝扩展以纳入持久化存储，从而构建三级层次结构。向量和图结构以与 CPU 内存中相同的格式存储在磁盘上，并通过按需加载将频繁访问的部分缓存到 CPU 和 GPU 层级中。该设计支持从数百万个向量扩展到数十亿个向量，同时针对不同工作负载保持一致的性能。

缓存管理器 位于 GPU 上，用于在 SANNS 操作期间协调这些存储层级之间的数据移动。它维护一个映射表，其中包含映射函数 $mapping(h_id) \rightarrow \{d_id \mid \text{NONE}\}$，该函数返回 GPU 缓存向量的设备标识符，或为驻留在 CPU 中的向量返回 *NONE*。该设计利用 GPU 内存的高带宽访问能力，实现大规模并行向量查找，同时避免 CPU-GPU 同步的开销。

索引构建。在索引构建期间，数据集会根据 GPU 内存容量进行分区。当扩展到采用持久化存储的大规模数据集时，SVFusion 从磁盘将每个分区的数据加载到 CPU，再将其传输到 GPU 以构建局部子图，最后将完成的子图写回磁盘。随后，CPU 在有界内存窗口内计算跨分区边，并合并这些子图，同时按需从磁盘加载向量。无论数据集规模如何，该方法都能确保内存使用情况保持稳定；最终图会持久化到磁盘，而频繁访问的部分则缓存于 CPU-GPU 层级中。

CPU-GPU 向量搜索。我们介绍 SVFusion，其搜索工作流如算法 1 所示。与以往的纯 GPU 方法不同，SVFusion 优先在图形处理器上执行缓存数据的计算，并动态将未缓存的计算卸载到中央处理器，或触发按需传输。并行搜索过程首先随机初始化每个查询的候选池（第 1–2 行）。随后在图形处理器上执行图遍历（第 5 行），直至候选池不再变化。随机入口点有利于图形处理器并行执行，并可避免动态更新期间种子维护产生偏差。在邻居扩展过程中（第 7–11 行），我们的缓存管理器会检查每个邻居向量的缓存状态（第 8 行），并将计算分配给中央处理器或图形处理器。命中缓存时，直接利用高带宽内存访问在图形处理器上处理向量。对于缓存未命中的情况，我们提出工作负载感知向量放置（WAVP）算法，以确定是将向量提升至图形处理器，还是直接在中央处理器上执行计算。这样便可将邻居划分为 X_i^{GPU} 和 X_i^{CPU}，由两个处理器并行计算距离（第 10–13 行）。这一自适应策略兼顾吞吐量与内存效率，使向量搜索在图形处理器内存约束下仍具备良好的可扩展性和响应能力。需要注意的是，在中央处理器和图形处理器之间维护数据副本，会在索引更新期间带来一致性挑战。我们通过立即同步逻辑删除的位集，并将图形更新从中央处理器批量传输至图形处理器来解决这一问题，详见第 5.3 节。

4.3 工作负载感知向量放置

为应对流式工作负载的动态访问模式，我们提出了一种自适应向量放置策略，以优化数据局部性和 ANNS 中的计算效率。

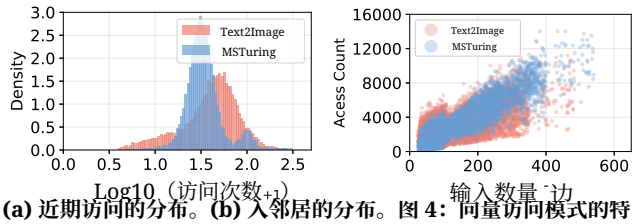
令 $X_t = \{x_1, x_2, \ldots\ldots, x_N\}$ 表示时间戳 t 时的完整向量集合。由于 GPU 内存约束，只有子集 $X_t^G \subseteq X_t$（其中 $|X_t^G| = M \ll N$）驻留在 GPU 内存中，其余向量则保存在 CPU 内存中。在访问向量 $x \notin X_t^G$ 时，**核心挑战**在于，CPU-GPU 协同处理需要为每次向量访问决定最优放置方式：(1) 将向量从 CPU 内存提升至 GPU 内存，或 (2) 直接在 CPU 上执行距离计算。这与传统的缓存替换策略有本质区别，例如 LRFU [37], ，后者旨在最大化缓存命中率。在 CPU–GPU 协同处理中，将每个未命中的向量都提升至 GPU 缓存可能导致过多的数据传输，而在 CPU 上计算所有查询则会造成 GPU 并行性利用不足。因此，我们的缓存策略联合优化传输成本和计算效率，自适应地决定是在 GPU 上缓存向量，还是直接在 CPU 上进行计算。

为了确定对于每个未驻留在 GPU 上的向量访问而言，这两种策略之间的最优选择，我们需要量化计算节省与数据传输开销之间的权衡。我们定义如下增益函数：

$$\text{gain}(x) = \lambda_x \cdot (T_{\text{CPU}} - T_{\text{GPU}}) - T_{\text{transfer}} \quad (1)$$

其中，λ_x 表示在指定时间窗口内对向量 x 的未来访问次数。T_{CPU} 和 T_{GPU} 分别表示在中央处理器和图形处理器上计算距离所需的平均时间。T_{transfer} 表示将一个向量传输至 GPU 内存的摊销成本，通常基于 2048 的批大小计算，以降低每个向量的通信开销 [9]。当 $\text{gain}(x) > 0$时，这意味着 GPU 加速带来的预期累计加速收益超过了数据传输成本，因此有理由将 x 提升至 GPU 内存中。

实现该增益函数的关键挑战在于 ANNS 期间获取 λ_x。由于未来访问频率在决策时本质上



(a) 近期访问的分布。(b) 入邻居的分布。图 4：向量访问模式的特征分析。

无法观测，我们开发了一个预测函数 F_λ，基于可观测的运行时信息来估计 λ_x。

预测函数设计。有效的 λ_x 预测需要具备与未来访问相关的特性。由于基于图的近似最近邻搜索依赖图遍历，访问模式自然取决于时间因素和图结构。因此，我们针对这两种特性，分析了在不同维度和数据类型的 MSTuring [4] 和 Text2Image [50] 数据集上进行近似最近邻搜索时的向量访问情况。图 4 展示了向量访问模式的主要特征。近期访问频率 $F_{recent}(x,\ t)$体现了时间局部性，但呈现的是中热分布，而非长尾分布。相比之下，入邻居度 $E_{in}(x)$与访问频率表现出很强的结构相关性，反映出搜索通常会遍历高连通性枢纽。然而，现有缓存策略（如 LRFU [37] ）基于近期性和频率做出替换决策，无法利用这种结构重要性。为了有效缓存图中的这些高连通性向量，我们的方法将图结构连通性明确纳入缓存决策过程。

基于这些观察，我们设计了一个预测函数，通过将时间局部性 $F_{recent}(x,\ t)$ 和结构连接性 $E_{in}(x)$ 作为关键因素相结合，来估计未来访问次数。鉴于线性组合方法简单且易于解释，我们采用了该方法；不过，也可以进一步探索其他拟合方法（例如多项式）：

$$F_\lambda(x) = \alpha \times F_{recent}(x,t) + \beta \times \log(1 + E_{in}(x)) \tag{2}$$

其中，参数 α 和 β 分别控制时间特征和结构特征的相对重要性。在实践中，我们固定 α 和 β 之间的比率，并根据性能评估调整其取值。缓存管理器使用滑动窗口机制维护 $F_{recent}(x,\ t)$，定期衰减较早的访问记录，并随着图拓扑的演变更新 $E_{in}(x)$。

理论分析。我们从理论上证明了预测函数能够有效指导缓存决策。我们将实际的未来访问次数 λ_x 建模为：

$$\lambda_x = \lambda_1 \cdot F_{recent}(x,t) + \lambda_2 \cdot \log(1 + E_{in}(x)) + \lambda_3 \tag{3}$$

其中 $\lambda_1, \lambda_2 > 0$ 为权重参数，λ_3 为偏置项。使用对数变换来减小 E_{in} x $\alpha = \lambda_1$ $\beta = \lambda_2$ λ_x =( ) 的极高 F_λ x $+\lambda_3$ $x > 0$ 值。通过设置 和 ，我们得到 ( ) 。当 gain( ) 时，缓存是有益的。当 $\lambda_x > \rho$ 时，该条件成立，其中 $\rho = \frac{T_{\text{transfer}} T_{\text{中央处理器}} - T_{\text{图形处理器}}}{\lambda_x = F_\lambda(x) + \lambda_3}$ 是传输开销与计算收益的成本比。由于我们有 ，该条件变为 $F_\lambda(x) + \lambda_3 > \rho$，这等价于 F_λ x $> \rho - \lambda_3$ $\tau = \rho - \lambda_3$ F_λ x $> \tau$ $\Rightarrow$ ( ) 。通过 $\lambda_x > \rho$ $\Rightarrow$ $x > 0$ 设置，我们得到 ( ) $\Leftarrow\Leftarrow$ gain( ) 。这表明，我们的方法将缓存决策简化为简单的阈值比较。

算法 2缓存未命中时的工作负载感知向量放置

输入： 不在 GPU 中的向量 x，GPU 内存中的向量子集 X_t^G，时钟指针 $clock$，引用位 ref
输出： 有效的设备标识符 d_id 或无

1: **如果** $F_\lambda(x) \le \theta$ **则**
2: **返回** 无
3: $F_{min} \leftarrow \min\{F_\lambda(x_i) \mid x_i \in X_t^G, ref[i] = 0\}$
4: **当真 执行**
5: $x_{curr} \leftarrow X_t^G[clock]$
6: **如果** $ref[clock] = 0$ **且** $F_\lambda(x_{curr}) = F_{min}$ **则**
7: 替换$(x_{curr},\ x)$
8: **返回** 映射(x_{curr})
9: **否则，如果** $ref[clock] = 1$ **则**
10: $ref[clock] \leftarrow 0$
11: $clock \leftarrow (clock + 1)$ 对 $|X_t^G|$ 取模

WAVP 算法。基于预测函数，我们设计了算法 2 所示的负载感知向量放置算法。对于当前不在 GPU 内存中的向量 x，该算法返回用于缓存的设备标识符，或返回 无 以表示基于 CPU 的执行。该放置算法包含两个阶段：选择性预取 和预测性替换。

选择性预取阶段（第 1–2 行）用于确定是否将向量 x 提升至 GPU 内存。我们初始化阈值$\theta = T_{\text{数据传输}} T_{\text{中央处理器}} - T_{\text{图形处理器}}$，该阈值对应于提升至 GPU 有益所需的最低频率。当 $F_\lambda(x) \le \theta$时，系统会将 x保留在中央处理器上进行原位计算。在预测性替换阶段（第 3–11 行），我们通过预测引导的驱逐策略扩展时钟扫描策略：当 GPU 内存已满时，系统会驱逐引用位为零且预测频率最低的向量 $F_\lambda(x)$，从而有效缓解动态 ANNS 工作负载下的抖动。该设计严格控制驱逐决策，以避免不必要的向量替换和冗余数据传输。

为扩展至大规模数据集，SVFusion将其内存层次结构扩展为包含磁盘存储，从而在图形处理器、中央处理器和磁盘之间形成级联查找流水线。发生缓存未命中时，系统在基于哈希的目录引导下，从较低层级异步预取向量，该目录用于跟踪数据驻留情况。当 CPU 内存饱和时，得分最低的向量会被降级至磁盘，从而确保高效的多层数据流和大规模场景下的一致性能。

4.4 SA 的实时协调 NNS

为同时支持高吞吐量和低延迟工作负载，我们设计了一种实时协调机制，可动态调整 GPU执行。尽管大批次能够提升吞吐量，但对延迟敏感的查询会通过低延迟执行路径立即处理，该路径结合了多流协调 [56] 和多 CTA 执行 [47]。自适应资源管理器还会在流式负载波动时进一步调整 GPU利用率，而冷启动处理则确保系统从初始化开始便保持稳定性能。

低延迟处理。为了在面向延迟的场景中实现小批量查询的低延迟，我们采用两种技术来利用 GPU 资源。首先，我们集成了 CUDA 多流协调机制，其中包含多个搜索流

以及一个专用更新流。此设计支持并发操作：搜索流并行处理不同批大小的查询，而更新流则异步处理插入和删除操作，并保证 §5.3 中详述的一致性。其次，受 CAGRA 的多 CTA 设计启发 [47]，系统支持查询内并行性，即多个 GPU 线程块协同处理单个查询。

自适应资源管理。 SVFusion 动态分配图形处理器资源，以适应不断变化的工作负载。为支持并发多流执行，我们将图形处理器内存划分为由轻量级自旋锁保护的独立分段，并通过标识符哈希将向量映射到各分段。系统会自动为工作负载开销较高的流分配更多图形处理器内存和计算资源。此外，缓存阈值 θ 会根据观测到的未命中率以及估计的未来访问次数 $F_\lambda(x)$ 的分布持续调整。当未命中率随着较高的平均 F_λ一同上升时，系统会提高 θ，通过提高替换选择的严格性来防止缓存抖动。所有调整均会自动进行，无需手动调优。

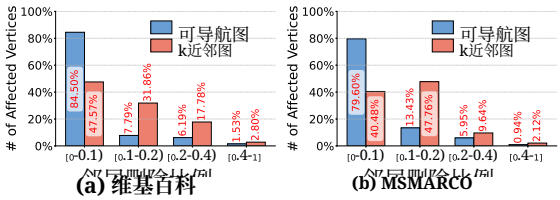
冷启动处理。 为避免系统开始处理流式操作时出现启动延迟，SVFusion 执行三阶段初始化：(1) **GPU 资源预热** 在图形处理器全局内存中预先分配向量存储空间，并创建用于异步执行的 CUDA 流；(2) **数据结构初始化**准备缓存管理器的映射表，并开始访问跟踪元数据；(3) **GPU 缓存预热** 根据预测的访问次数预加载向量。对于大型数据集，SVFusion 会缓存根据公式 2 预测的排名靠前的高入度向量，以最大限度地减少 GPU 缓存未命中。

5 索引更新

5.1 向量插入

插入过程包括两个阶段：(1) 执行搜索操作，为新向量寻找候选邻居；(2) 随后从新向量向选定的邻居建立边，并为这些邻居更新反向边。为实现高吞吐量，我们的框架利用图形处理器的大规模并行性，支持同时批量插入大量向量；相比之下，受线程池限制，基于 CPU 的方法一次只能处理约数百个向量 [59]。对于低延迟要求，我们采用 §4.4 中介绍的技术，例如多流并采用多 CTA，以保持较高的 GPU利用率。

我们进一步设计了一种基于排名的候选项重排序策略。对于候选项列表中顺序位置为 i 的每个候选项，我们统计此前选中的邻居（位于位置 $j < i$）的邻居列表中包含该候选项的数量——每次出现都代表一条可绕行路径，可以绕过直接边。随后，我们按可绕行路径数量升序排列候选项，并选取排名靠前、数量不超过图度数的候选项。该方法避免了开销高昂的距离计算，遍历候选项集合 C 的复杂度仅为 O(|C| log |C|)。在反向边插入过程中，我们采用细粒度原子操作和线程局部缓冲区来处理并发图更新；与粗粒度锁定相比，这种方法在保持一致性的同时具有更好的可扩展性。



(a) 维基百科　(b) MSMARCO

图 5：已删除邻居数量的分布。

5.2 向量删除

SVFusion 通过三个阶段处理向量删除：(1) 对非活跃顶点进行逻辑标记，以避免干扰查询；(2) 进行局部拓扑感知修复，以快速恢复连通性；(3) 定期异步执行全局合并。

5.2.1 **逻辑删除。** SVFusion 采用轻量级的惰性删除策略，以高效支持高频删除操作。SVFusion 不会立即修改图结构，而是使用位集将已删除顶点标记为逻辑删除。在后续的插入、删除和查询过程中，系统会自动跳过这些已标记的顶点。通过推迟代价高昂的结构修复和邻居更新操作，该策略最大限度地降低了删除延迟，并避免了同步瓶颈。

5.2.2 **受影响顶点修复。** 虽然逻辑删除能够实现低延迟删除，但随着逻辑删除不断累积，图结构会逐渐碎片化，导致连通性和搜索效率下降。以往的删除处理方法 [53]主要针对相对邻域图（RNG）[20],进行设计，其中删除操作主要影响高度数枢纽，而全局整合会推迟到达到删除阈值（例如 20%）后进行。然而，K近邻图（KNNG）采用统一的度数约束，导致删除影响均匀地扩散到所有顶点。我们的分析（图 5）表明，删除 10% 的顶点后，KNNG 中拥有 10–40% 已删除邻居的顶点数量增加了 3.26×，拥有超过 40% 已删除邻居的顶点数量则增加了 2.04× 。这种普遍存在的连通性损失会在达到全局修复阈值之前很久就降低搜索质量，因此，随着删除不断累积，必须持续进行增量修复，并优先处理受影响最严重的顶点。

局部拓扑感知修复。 SVFusion 通过局部拓扑感知修复策略解决性能快速退化的问题。该方法不会立即清理已删除的顶点，而是利用删除位集快速识别受影响严重的顶点 V^L（其邻居中超过 50% 已被删除），并仅对这些关键顶点进行轻量级修复。给定一个顶点 $v \in V^L$，与针对每个已删除邻居 p都将 N_{out}（p）中的所有顶点连接到 v 这种代价高昂的合并操作不同，我们的轻量级修复至多从 N_{out}（p）中选择 c 个顶点连接到 v。常数 c 较小（我们采用$c = 8$），因此新增边的数量为 O（cR），远小于合并操作期间的 O（R^2），其中 R 为图的度数。与对所有受影响顶点执行代价高昂的修复相比，这种增量式方法只需极小的开销，便能保持图的质量。

全局合并。 当删除比例超过预定义阈值（例如 20%）时，SVFusion 会通过汇总已删除顶点的出邻居中的候选项，对所有受影响邻域执行全局合并。此合并在最新图

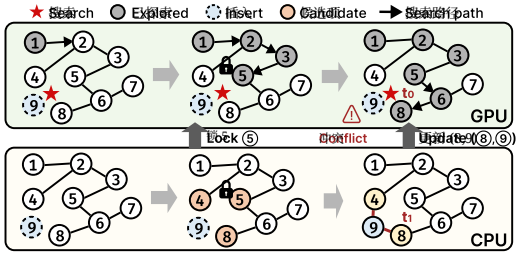


图6：并发控制工作流程示意图。

快照上异步地在后台执行，使插入和查询能够不受干扰地并发进行。为确保与顺序操作的一致性，该过程利用图快照和合并机制来协调不同图版本之间的更新。

5.3 并发控制

SVFusion 采用两级并发控制协议，以支持在分层内存层级上安全且高效地执行 SANNS 操作。我们首先通过细粒度锁定和同步数据结构建立一致性保证，确保即使更新交错执行，搜索仍能观察到一致的数据。在此基础上，多版本机制通过快照隔离将开销高昂的后台整合与串行操作解耦，从而实现并发执行而不会发生阻塞。

一致性保证。SVFusion 在 GPU、CPU 和磁盘层级之间提供一致的索引视图，确保所有查询和更新都能观察到分层索引的连贯状态。一致性通过细粒度的本地同步以及跨三个共享结构协调执行的跨层更新来维护：驻留于中央处理器的图、多层删除位集，以及中央处理器—图形处理器映射表。

本地同步。在每个内存层级内，细粒度锁协调并发操作。搜索在读锁下遍历邻居列表，而插入操作在候选搜索时获取读锁，并仅在边建立期间对受影响的邻居列表升级为写锁。删除操作在排他锁下更新共享位集，同时标记受影响的版本，从而允许其他操作跳过逻辑上已删除的顶点。

跨层协调。为在更新期间维持一致性，SVFusion 采用中央处理器和图形处理器内存之间的两阶段同步协议。对于删除，在排他锁保护下跨层原子更新位集，以确保可见性一致。对于拓扑更新（插入、修复或合并），首先在中央处理器上通过写锁提交已修改的邻居列表，并递增版本号；与此同时，对应的 GPU 缓存条目保留其版本。随后，将这些已更新的列表分批异步传播至图形处理器，并在传输完成后更新版本标识符。在这一暂态阶段，图形处理器查询若遇到版本冲突，则会透明地回退至 CPU 内存，以确保结果一致。基于这一轻量级版本控制机制，异步更新绝不会覆盖并发修改的区域，从而保证正确性并避免同步停滞。这种设计在支持跨分层内存层级进行高吞吐量、非阻塞传播的同时，保证了并发操作下的正确性。图6展示了 SVFusion 中并发插入与搜索

表2：实验数据集统计信息。

数据集	N	大小（GB）	D	N_q	Type
维基百科 [8]	35,000,000	101	768	5,000	Text
MSMARCO [5]	101,070,374	293	768	9,376	Text
MSTuring [4]	200,000,000	75	100	10,000	Text
Deep1B [16]	1,000,000,000	358	96	10,000	图像
Text2Image [50]	100,000,000	75	200	10,000	图像

通过细粒度锁定和跨层同步，确保新向量正确可见。

多版本机制。为了将开销高昂的后台整合与在线向量操作解耦，SVFusion采用多版本并发机制，在不同内存层级中维护一小组一致的图版本。每个版本都提供图拓扑的完整、不可变快照，以供并发后台处理；与此同时，新操作则在活动版本上进行，并满足一致性保证。

每当后台任务（例如整合或大规模修复）启动时，都会创建一个新图版本。系统将当前图元数据复制为只读快照 G_{t_0}，供任务执行。插入和删除操作则继续在活动图 G_{t_1} 上进行。后台任务完成后，会生成一个已更新的图 G'_{t_0}，并将其合并回 G_{t_1}，形成统一版本。为避免版本无限累积，SVFusion采用有界版本策略：达到上限后推迟创建新版本，并按顺序安排整合。

增量子图追加。在合并过程中，我们首先识别快照创建后插入的新顶点：V V V_{new} (G_{t_1} G_{t_0} G_{new} =) \ ( )。随后将 $G_{t_1}[V_{new}]$ G'_{t_0} 对应的诱导子图直接追加到整合后的结果中。这种增量方法无需重新处理整个图，就能确保最近插入的顶点保留在合并后的结构中，从而显著降低合并延迟。

反向边集成。为维持双向连通性，我们记录整合窗口 $[t_0, t_1]$ 内现有顶点与新插入顶点之间发生的反向边更新。每项更新都记录为一个三元组（v，v_{new}，d），其中 $v \in V(G_{t_0})$、$v_{new} \in V_{new}$ 和 d 表示它们的预计算距离。

在提交新版本之前，SVFusion 会以原子方式应用这些更新，将相应的反向边添加到每个受影响顶点的邻域中。这种两阶段策略可确保合并索引的结构完整性并维持其连通性。

6 评估

6.1 实验设置

硬件配置。 实验在一台运行 Ubuntu 18.04.6 LTS 的双路服务器上进行，该服务器配备两颗 Intel Xeon Gold 5218 中央处理器（32 个物理核心、64 个线程，组成两个 NUMA 节点）、376 GB DDR4 DRAM（12 个通道，256 GB/s）、一块配备 40 GB 设备内存的 NVIDIA A100 图形处理器（PCIe 3.0），以及用于持久化存储的 Intel D3-S4510 960 GB 企业级固态硬盘。

数据集。 我们在五个广泛使用的真实世界数据集上评估 SVFusion。这些数据集具有多样的维度和数据类型，并已被广泛应用于 [13, 44, 52]。表2总结了每个数据集的详细属性。查询从每个数据集中随机采样，并通过穷举线性扫描计算真实值。

工作负载。 我们的评估涵盖多种流式工作负载模式，每种模式均由一组向量和一个在 §2.2 中定义的操作序列组成。这些工作负载在以下方面有所不同：(i) 插入与删除之间的间隔；(ii) 单步删除的向量之间的空间相关性。为模拟真实的流式场景，我们以不同的 QPS 速率持续从 Kafka 源消费搜索、插入和删除向量。我们采用了来自 *2023* 年 *Big ANN* 挑战赛 [52]和 *MSTuring-IH* 生成于 [44]的三种代表性工作负载模式：

• **滑动窗口**：数据集均匀划分为 $T_{max} = 200$ 个分段。每步插入一个分段；从第 T_{max} 2 + 1 T_{max} 2 / 步开始，删除在 / 步之前插入的分段。评估从 $T_{max}/2 + 1$ 开始。此工作负载模拟一个仅保留最新数据的滑动窗口。

• **ExpirationTime**：为向量分配不同的生命周期——短期（10 步）、长期（50 步）和永久（100 步）——其比率为 10:2:1。每步插入数据集的 $1/T_{max}$，并相应删除过期向量。此设置用于评估索引在异构数据生命周期下的行为。

• **Clustered**：使用 k-means 将数据集划分为 64 个簇，迭代 5 轮。每轮交替执行插入和删除阶段：从每个簇中随机插入向量，然后删除向量。由于空间相邻点会被同时修改，此工作负载会造成显著的分布偏移。

• **MSTuring-IH**：从 2,000 万个向量开始，在 1,000 次操作中，数据集增长至 2 亿个向量，插入比率为 90%，搜索比率为 10%，以测试大规模增量增长下的可扩展性和查询稳定性。

实现与基线方法。 我们按照图 2 所示的架构扩展 cuVS 24.02 库，实现了 SVFusion [3] 。我们将 SVFusion 与以下基线方法进行比较：

• **HNSW** [42] 是一种具有代表性的基于图的近似最近邻搜索方法，在工业系统中得到广泛部署，并受到学术界的深入研究。我们使用 $M = 48$ 和 $ef_construction = ef_search = 128$。

• **FreshDiskANN** [53] 是最先进的基于图的近似最近邻搜索方法，可支持在内存或固态硬盘上的动态场景。我们按照 [2] $R = 64$ $l_b = l_s =$ 中的设置配置 128 $\alpha = 1.2$ $l_d = 128$ FreshDiskANN：„用于插入，且用于搜索。

• **CAGRA** [47] 是 NVIDIA 面向 GPU 设计的高性能近似最近邻搜索算法。它能够实现卓越的性能，但在大规模数据集上会受到 GPU 内存限制。

• **PilotANN** [26] 是一种近期提出的混合架构，能够有效实现 GPU 加速，但不支持动态更新。

• **GGNN** [25] 是一种基于 GPU 的近似最近邻搜索方法，构建了类似 HNSW 的分层图结构，但其设计面向能够容纳在 GPU 内存容量中的静态数据集。

现有的基于 GPU 的方法要么仅支持静态构建 [47, 68, 69] ，要么受到 GPU 内存 [56]限制，动态能力有限。为在流式工作负载下进行公平比较，我们通过将距离计算卸载到 GPU，创建了 FreshDiskANN 和 HNSW 的 GPU 加速版本，并通过 GPU 缓存管理扩展 GGNN 以支持更新。

评估指标。 我们关注四项关键指标：**(1) 吞吐量**。它衡量流式工作负载下每秒处理的搜索查询和更新操作（插入/删除）数量；**(2) 延迟**。它表示从提交请求到返回最终结果所经历的端到端时间；（3）**召回率**。召回率@k 表示每个搜索查询的召回率，默认为 k=10 ； **(4) GPU 缓存未命中率**。它定义为 ANNS 期间需要从 CPU 内存而非 GPU 缓存中获取数据的向量访问所占的百分比。较低的未命中率表示频繁访问的向量具有更好的缓存驻留。

6.2 SANNS 总体性能

为评估 SVFusion 的总体性能，图7展示了五个数据集上七种工作负载的召回率、搜索吞吐量、插入吞吐量和 GPU 未命中率。这些工作负载包含大量更新，性能目标是在 900 毫秒 [39] 内实现高吞吐量和有界的 p99 延迟。由于所有方法都使用逻辑标记来延迟删除，因此瞬时速率无法进行比较，故未纳入删除吞吐量。总体而言，SVFusion 始终优于所有基线方法：与 FreshDiskANN 相比，其搜索吞吐量最高提升 9.5×，插入吞吐量最高提升 71.8×，同时始终保持最高召回率。

召回率分析。 SVFusion 在所有流式工作负载中始终保持较高的搜索质量，Recall@10 得分在 91% 至 96% 之间，与所有基线方法相当或更优。其召回率平均比 FreshDiskANN 高 0.4–3.4%，这得益于其**局部拓扑感知修复机制** 能够主动检测并恢复关键图连通性，从而减轻随时间推移产生的结构退化。相比之下，HNSW 缺少删除修复策略，导致图碎片化持续存在，因此召回率最低。在持续进行结构更新时，所有方法的召回率均会波动，尤其是在聚类工作负载中。

吞吐量分析。 图7表明，SVFusion 在搜索和插入操作中均实现了最高吞吐量，其搜索吞吐量平均比 FreshDiskANN 和 HNSW 高 20.9×，插入吞吐量高 3.5×。值得注意的是，当数据集超出图形处理器容量后，GGNN 等 GPU加速基线方法的运行速度比 SVFusion 慢 4.5–7.8×，因为频繁的 CPU–GPU 数据传输抵消了图形处理器加速带来的优势。SVFusion 通过自适应缓存算法维持高性能。在从空数据集初始化的工作负载（*Expiration Time*、*Clustered* 和*MSTuring-IH*）的早期时间步中，由于图形处理器计算资源得到充分利用且缓存未命中极少（<1%），搜索吞吐量达到峰值 40K。

未命中率分析。我们还报告了插入和搜索操作的 GPU 缓存未命中率，以更好地刻画多层内存的行为。尽管 GPU 内存仅占数据集的一小部分，SVFusion 在动态工作负载下仍能持续保持较低的未命中率，这表明其缓存放置策略能够有效捕捉时间和图结构访问模式，并最大限度地减少冗余驱逐。

延迟分析。 为全面评估 SVFusion 在不同流式处理强度下的延迟性能，我们在 MSTuring 数据集上将请求速率从低负载（每秒查询数=500）提升至高负载（每秒查询数=10,000）。图 8 报告了搜索和插入操作的第50百分位数、第95百分位数和第99百分位数延迟。在低请求速率下，SVFusion 的延迟与基线系统相当；由于在查询并发度有限时，CPU–GPU 同步开销无法得到有效摊销，其第99百分位数延迟适度增加了 15–30%。

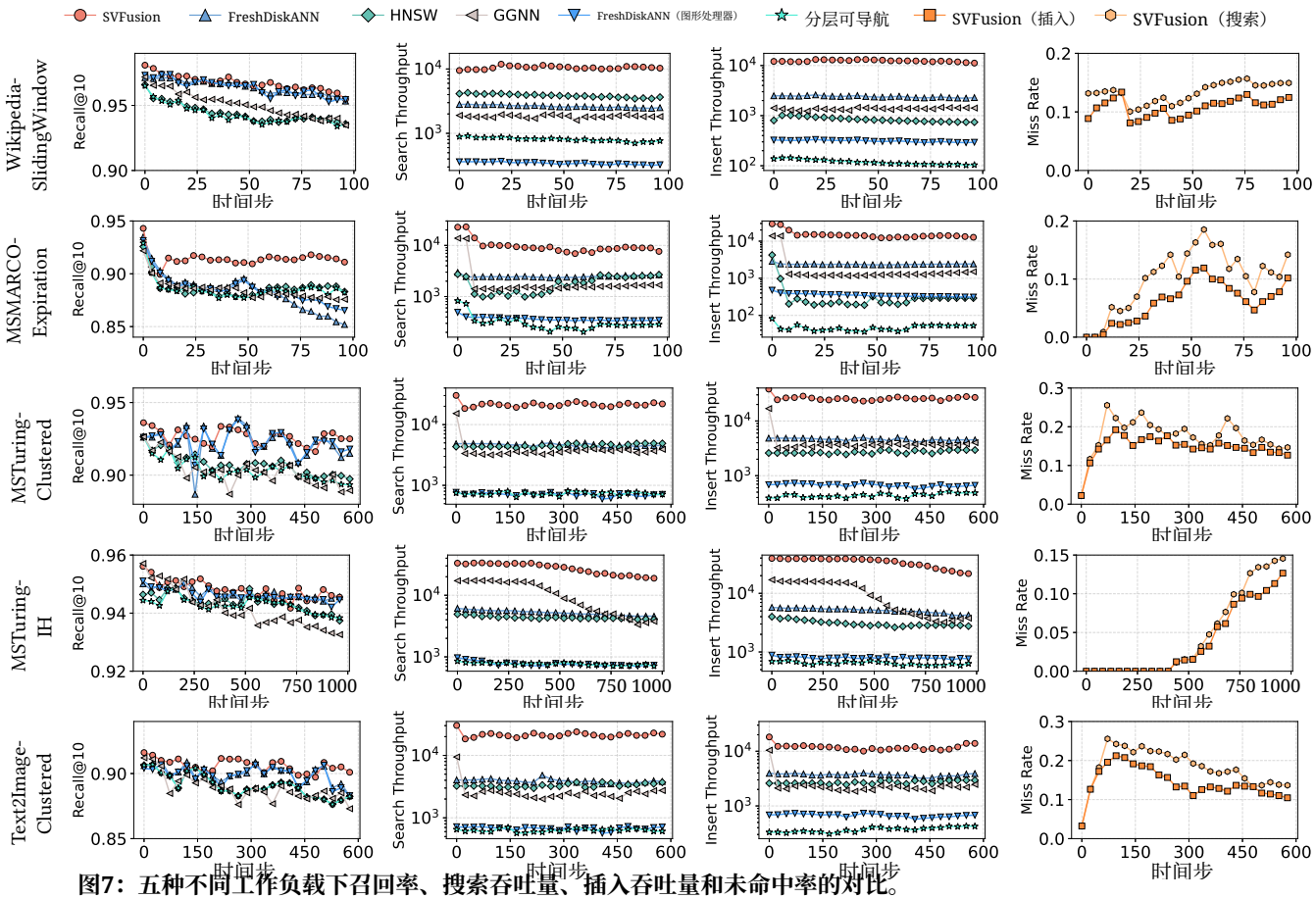


图7：五种不同工作负载下召回率、搜索吞吐量、插入吞吐量和未命中率的对比。

尽管如此，所有观测到的延迟均低于 10 毫秒，表明其对响应能力的实际影响极小。在中等吞吐量下，SVFusion 展现出较强的延迟稳定性：搜索延迟的第50百分位数为 4.3 毫秒，第99百分位数为 7.9 毫秒；与 FreshDiskANN 和 HNSW 相比，分别降低了 32% 和 18%。这一改进得益于 SVFusion 的多流协调机制，该机制通过重叠计算与数据传输，减轻并发操作之间的阻塞。在高请求速率下，这一优势更加明显：基线方法出现严重的队列积压，第99百分位数延迟超过 900 毫秒（慢 45–50.7×），而 SVFusion 将搜索和插入的尾延迟分别维持在 16.5 毫秒和 45.3 毫秒。这些结果凸显了 SVFusion 可扩展的协调设计：在实现高吞吐量的同时保持持续较低的延迟；相比之下，由于并行性有限且存在资源争用，CPU受限基线在高负载下会迅速达到饱和。

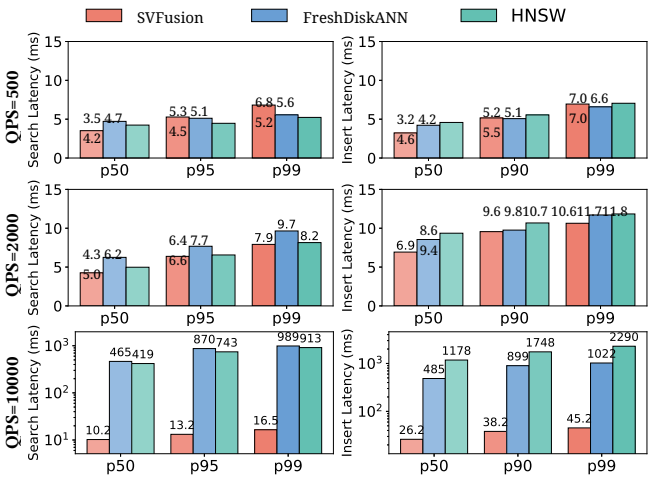


图 8：第50、第95和第99百分位数的查询延迟对比。

6.3 组件分析

自适应向量缓存的有效性。我们通过两组互补实验，使用四种替代方案评估我们的缓存策略：不含*WAVP*的*SVFusion*，该方案始终在中央处理器上为缓存未命中计算距离，且不将向量传输到图形处理器；*LRU/LFU/LRFU*，这些方案以传统缓存策略取代我们的 WAVP 机制。

替换策略。我们首先在滑动窗口工作负载下评估工作负载感知向量放置（WAVP）策略的有效性。图9显示，采用WAVP的SVFusion吞吐量最高、延迟最低，性能优于SVFusion（不含WAVP）、LRFU、最不经常使用和最近最少使用，吞吐量最高提升7.2×，延迟最高降低5.1×。这些收益

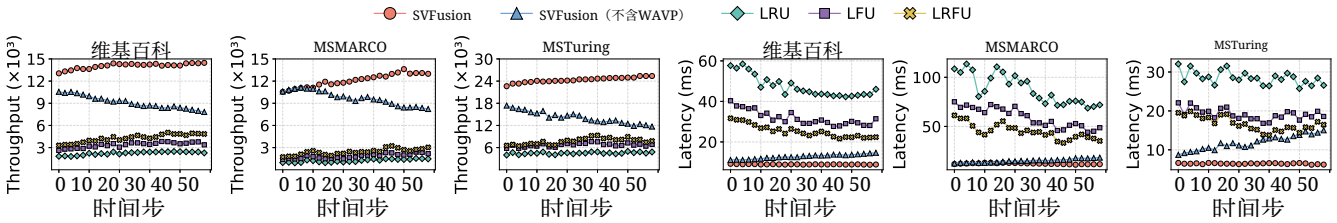


图9：不同替换策略对更新吞吐量和延迟的比较。

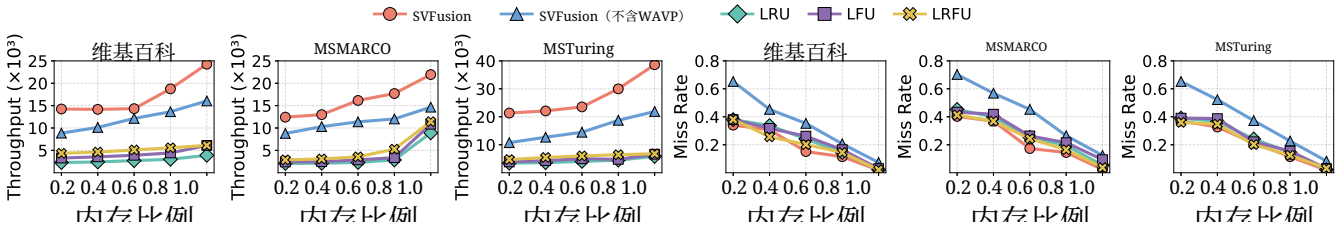


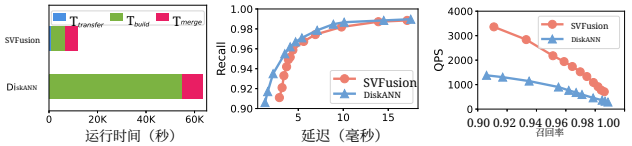
图10：不同 GPU 内存比例对更新吞吐量和未命中率的影响。

源于传统缓存策略忽视了工作负载动态变化，导致向量替换过于频繁，并产生高昂的CPU–GPU数据传输开销。WAVP通过自适应地建模时间和结构访问模式，最大限度地减少不必要的数据迁移，并最大限度地提高图形处理器利用率，从而在不同工作负载下实现高效率和稳定的性能。

内存约束。为评估我们的缓存策略在不同图形处理器内存条件下的稳健性，我们使用相同的数据集，并将可用图形处理器内存从20%扩展至100%。如图10所示，我们的方法在所有设置下均持续优于所有基线方法，其平均吞吐量分别比SVFusion（不含WAVP）、LRFU、最不经常使用和最近最少使用高出1.7×、3.3×、4.1×和5.6×。值得注意的是，即使在图形处理器容量降低、缓存未命中率上升的情况下，SVFusion仍能保持稳定的吞吐量。这表明，我们的设计能够有效地将中央处理器计算与数据传输重叠，从而隐藏内存延迟，并在图形处理器内存受限的情况下维持高性能。

磁盘层的有效性。 我们在Deep1B数据集 [16]上评估了磁盘扩展的SVFusion框架。如图11a所示，与DiskANN相比，SVFusion的索引构建速度快5.26×，这主要归因于基于图形处理器的子图构建速度提升了9.1×。然而，磁盘合并阶段仍然是流水线中的主要瓶颈。图11b和11c展示了不同召回率水平下的吞吐量和延迟性能。在高召回率目标（≥ 0.95）下，SVFusion的延迟比DiskANN高0.7–1.6%。这一轻微下降源于有限的图形处理器内存所导致的更多数据分区，这在一定程度上削弱了跨分区连通性。然而，在相当的召回率水平下，SVFusion的吞吐量比DiskANN高2.3×，凸显了我们的多层次分层索引设计的有效性。

删除策略的影响。 我们比较了三种方法：仅采用惰性删除、采用惰性删除并进行全局合并，以及采用我们结合局部修复的方法。如图12所示，与前两种方法相比，我们的方法分别将召回率提高了5.2%和2.3%，同时将开销降低了57.6%。与仅进行全局合并相比。



（a）构建时间 （b）召回延迟 （c）召回吞吐量

图11：GPU-CPU-磁盘框架的影响

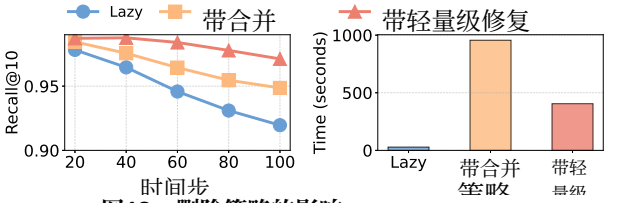


图12：删除策略的影响。

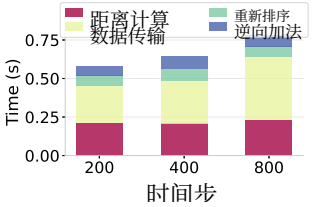


图13：性能分解。

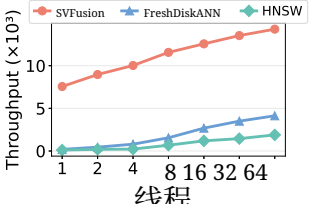


图 14：线程数的影响。

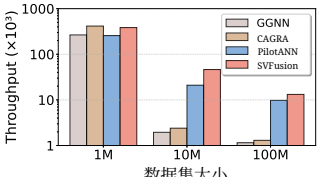


图 15：GPU方法。

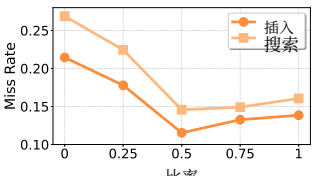


图16：预测参数。

性能分解与可扩展性。我们将 SVFusion 的插入耗时分解为几个关键组成部分（图13）：数据传输（45.4%）、距离计算（33.6%）、启发式重排序（10.3%）和逆向添加（10.6%）。这揭示了两点：(1) 尽管采用了 GPU 加速，随着索引大小增长，CPU–GPU 数据移动仍是主要瓶颈；(2) 距离

表 3：一致性保证的影响。

\ 每秒查询数	Recall@1				p99 延迟（毫秒）			
	500	2K	5K	10K	500	2K	5K	10K
同步	0.96	0.96	0.96	0.96	4.3	6.7	10.8	33.2
不同步	0.96	0.74	0.51	0.18	4.3	4.8	5.5	7.4

计算保持稳定，证明了我们采用流水线的 CPU–GPU 并行策略的有效性。我们还通过将线程数从 1 调整至 64 来评估可扩展性（图 14）。虽然所有方法的吞吐量都随着线程数增加而提升，但超过 16 个线程后，收益逐渐减小，平均仅为 1.24×。

CPU-GPU 向量搜索的有效性。 图 15 展示了在 Recall@10=0.9 下，针对不同数据集规模，CPU-GPU 在 MSMARCO 上的搜索性能。对于能够放入图形处理器内存的数据集，SVFusion 的性能与 CAGRA 相当，但由于需要维护 CPU 端数据结构，吞吐量略低。当数据集超出图形处理器内存容量时，CAGRA 和 GGNN 的性能显著下降，这是因为通过统一虚拟内存（UVM）进行的 CPU-GPU 数据传输效率低下。与 PilotANN 相比，SVFusion 在所有规模下均实现了 1.5×-2.2× 倍加速，这是因为 PilotANN 的两阶段设计需要进行 CPU 端精化。这表明，我们的 CPU-GPU 协同处理方法无需数据压缩即可高效处理大规模向量搜索。

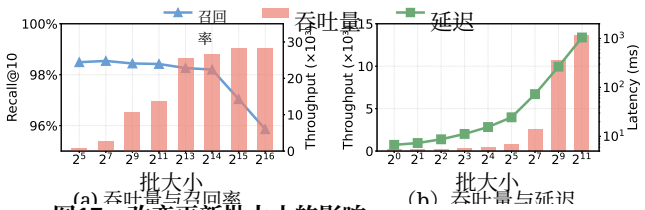
一致性保证的有效性。 为验证 §5.3 中的并发控制协议，我们设计了一项压力测试：每个批次都交错包含 50% 的插入操作和 50% 的搜索查询，批大小为 10 个向量，请求速率从 500 到 10K 每秒查询数不等。我们采用 Recall@1 作为主要指标，因为正确的系统应始终将新插入的向量作为最近邻返回。如表 3 所示，启用同步的 SVFusion 在所有请求速率下都能将 Recall@1 稳定保持在 0.96，证实了其具有强大的读后写一致性。相比之下，无同步 变体的性能严重下降：Recall@1 从每秒查询数为 500 时的 0.96 降至每秒查询数为 10K 时的 0.18（下降 81%），因为搜索越来越频繁地观察到部分更新或不一致的图状态。虽然强制同步会使最高负载下的 p99 延迟从 7.4 毫秒增加到 33.2 毫秒，但这一开销是有限的，且合情合理，因为它能确保在极端流式工作负载下的正确性。这些结果表明，我们的细粒度多层协调协议能够有效平衡

高吞吐量 SANNS 场景下的一致性和性能。

6.4 参数敏感性

预测参数。 我们通过改变比率 $\frac{\alpha}{\alpha+\beta}$，评估预测函数中的参数对未命中率的影响。其中，0 表示仅根据图拓扑进行预测，1 表示根据近期访问进行预测。图16显示，提高近期访问的权重通常会降低未命中率；当比率约为 0.6 时，性能最佳。

批大小。 我们在图17中分析了批大小对 SVFusion 性能的影响。较大的批次会显著提升吞吐量，但由于图更新延迟，召回率会在超过 2^{13} 后下降。延迟也会呈指数级增长，在 2^{11}, 时超过一秒，这体现了经典的延迟–吞吐量权衡。SVFusion



(a) 吞吐量与召回率　(b) 吞吐量与延迟

图17：改变更新批大小的影响。

支持自适应批大小调整，以动态平衡流式 ANNS 工作负载的吞吐量和查询延迟。

7 相关工作

向量索引。 近似最近邻搜索是一项经典问题，已有大量研究。基于分区的方法 [15, 22, 24, 28, 33, 41, 44, 66]利用树结构和量化将向量组织为空间分区，但会受到维度灾难的影响。基于哈希的方法 [21, 30, 58, 70] 将高维向量映射到低维哈希桶，以加速相似性搜索。基于图的方法 [14, 20, 32, 49, 64, 71] 构建邻近图，从而实现高效的向量搜索。然而，大多数技术针对静态数据集，在频繁更新时表现不佳。

流式向量搜索。 近期研究着重于在向量搜索系统中实现动态更新，以支持流式场景。FreshDiskANN [53] 会将新向量保存在内存中，然后定期将其合并到驻盘图中。SPFresh [63] 采用基于聚类的方法，并通过轻量级增量再平衡，在数据分布发生变化时调整分区。其他方法 [56, 61, 62] 则着重于维护图连通性或实现并行加速。尽管取得了这些进展，现有系统仍面临显著局限：它们要么仅支持有限的数据集规模，要么只能处理特定操作（插入），要么在高维空间中的搜索质量会下降。

GPU 加速的近似最近邻搜索。 利用 GPU 进行向量搜索，以应对大规模场景下的计算需求，已得到广泛研究。Faiss [35] 和 RAFT [1] 率先推动了从中央处理器到图形处理器的迁移，而 SONG [69],、GANNS [65], 和 CAGRA [47] 则通过算法与内核优化提升了 GPU 并行度。在系统设计优化方面， [25, 26, 36, 54, 68] 逐渐形成了整合异构 CPU-GPU 调度、在 GPU 与 CPU 之间分配数据，以及建立定期同步机制的方案。然而，这些方法针对的是静态数据集，在查询期间索引保持不变。

8 结论

我们提出了 SVFusion，这是一种面向大规模实时向量搜索的 GPU-CPU-磁盘协同框架。它引入了一个跨越多个内存层级的分层向量索引，可在异构硬件间高效分配数据。我们采用自适应缓存机制，通过综合考虑访问模式和图结构关系来优化数据放置。SVFusion 还采用并发控制协议，在确保数据一致性的同时保持高吞吐量和低延迟。实验结果表明，与各种流式工作负载下的最先进方法相比，SVFusion 在保持更高召回率的同时，吞吐量最高提升 71.8×，延迟降低 1.3× 至 50.7× 。

参考文献

[1] 2022. Rapidsai/raft：RAFT 包含广泛使用的数据科学、图和机器学习基本算法与基本组件。https://github.com/ rapidsai/raft. [2] 2023. NeurIPS 2023 流式处理挑战赛及后续内容。https://github.com/harsha-simhadri/big-ann-benchmarks/tree/main/neurips23/streaming. [3] 2024. cuVS：GPU 上的向量搜索与聚类。https://github.com/rapidsai/cuvs/tree/v24.12.00. [4] 2024. Microsoft Turing-ANNS-1B。https://learning2hash.github.io/publications/ microsoftturinganns1B/. [5] 2024. microsoft/ms_marco。https://huggingface.co/datasets/microsoft/ms_ marco.[6] 2024. 新的嵌入模型和 API 更新。https://openai.com/index/new-embedding-models-and-api-updates/. [7] 2024. 更新索引。https://github.com/rapidsai/cuvs/issues/295. [8] 2024. wikimedia/wikipedia。https://huggingface.co/datasets/wikimedia/ wikipedia. [9] 2025. CUDA C++ 最佳实践指南。https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html. [10] 2025. Gemini 提供的个性化帮助。https://gemini.google/overview/ personalization. [11] 2025. 有状态智能体构建平台。https://www.letta.com/. [12] 2025. AlloyDB 的 ScaNN：首个适用于数百万至数十亿向量的 PostgreSQL 向量搜索索引。https://cloud.google.com/blog/products/databases/how-scann-for-alloydb-vector-search-compares-to-pgvector-hnsw. [13] Martin Aumüller、Erik Bernhardsson 和 Alexander Faithfull。2020年。ANN-Benchmarks：近似最近邻算法的基准测试工具。Information Systems 87 (2020), 101374. [14] Ilias Azizi、Karima Echihabi 和 Themis Palpanas。2023年。Elpis：面向可扩展数据科学的基于图的相似性搜索。VLDB基金会会刊 16, 6 (2023), 1548–1559. [15] Artem Babenko 和 Victor Lempitsky。2014年。倒排多重索引。IEEE模式分析与机器智能汇刊（TPAMI）。37, 6 (2014), 1247–1260.[16] Artem Babenko 和 Victor Lempitsky。2016年。深度描述符十亿级数据集的高效索引。载于 IEEE计算机视觉与模式识别会议论文集。2055–2063. [17] Kenneth L. Clarkson。1994年。近似最近点查询算法。载于计算几何研讨会论文集。160–164. [18] Wei Dong、Charikar Moses 和 Kai Li。2011年。针对通用相似性度量的高效 k 近邻图构建。载于 WWW。577–586. [19] Cong Fu、Changxu Wang 和 Deng Cai。2022年。利用卫星系统图进行高维相似性搜索：效率、可扩展性和未索引查询兼容性。IEEE模式分析与机器智能汇刊（TPAMI）44, 8 (2022), 4139–4150.[20] Cong Fu、Chao Xiang、Changxu Wang 和 Deng Cai。2019年。利用导航扩展图进行快速近似最近邻搜索。VLDB基金会会刊 12, 5 (2019), 461–474. [21] Junhao Gan、Jianlin Feng、Qiong Fang 和 Wilfred Ng。2012年。基于动态碰撞计数的局部敏感哈希方案。载于国际数据管理会议（SIGMOD）论文集。541–552.[22] Jianyang Gao、Yutong Gou、Yuexuan Xu、Yongyi Yang、Cheng Long 和 Raymond Chi-Wing Wong。2025年。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), 1–26. [23] Jianyang Gao 和 Cheng Long。2023年。高维近似最近邻搜索：可靠且高效的距离比较操作。ACM数据管理会议论文集 1, 2 (2023). [24] Jianyang Gao 和 Cheng Long。2024年。RaBitQ：利用理论误差界量化高维向量以进行近似最近邻搜索。ACM数据管理会议论文集 2, 3 (2024), 167. [25] Fabian Groh、Lukas Ruppert、Patrick Wieschollek 和 Hendrik P. A. Lensch。2023年。GGNN：基于图的GPU最近邻搜索。IEEE大数据汇刊 9, 1 (2023), 267–279. [26] Yuntao Gui、Peiqi Yin、Xiao Yan、Chaorui Zhang、Weixi Zhang 和 James Cheng。2025年。PilotANN：内存受限的GPU加速向量搜索。CoRR abs/2503.21206 (2025).[27] Rentong Guo、Xiaofan Luan、Long Xiang、Xiao Yan、Xiaomeng Yi、Jigao Luo、Qianya Cheng、Weizhi Xu、Jiarui Luo、Frank Liu、Zhenshan Cao、Yanliang Qiao、Ting Wang、Bo Tang 和 Charles Xie。2022年。Manu：云原生向量数据库管理系统。VLDB基金会会刊 15, 12 (2022年8月), 3548–3561.[28] Guoyu Hu、Shaofeng Cai、Tien Tuan Anh Dinh、Zhongle Xie、Cong Yue、Gang Chen 和 Beng Chin Ooi。2025年。HAKES：面向嵌入搜索服务的可扩展向量数据库。VLDB基金会会刊 18, 9 (2025), 3049–3062. [29] Jui-Ting Huang、Ashish Sharma、Shuying Sun、Li Xia、David Zhang、Philip Pronin、Janani Padmanabhan、Giuseppe Ottaviano 和 Linjun Yang。2020年。Facebook搜索中的基于嵌入的检索。载于 ACM 国际会议[发现]与数据挖掘会议（KDD）。2553–2561. [30] Qian Li、Kaixiang Yang、Cheng Xu、Jianliang Xu、Yikai Zhang、Qiong Fang 和 Wilfred Ng。2015。面向近似最近邻搜索的查询感知局部敏感哈希。VLDB基金会会刊 9, 1 (2015), 1–12。[31] Masajiro Iwasaki。2016。用于邻近搜索的剪枝双向K近邻图。载于相似性搜索与应用国际会议，第 9939 卷。20–33。[32] Suhas Jayaram Subramanya、Fnu Devvrit、Harsha Vardhan Simhadri、Ravishankar Krishnawamy 和 Rohan Kadekodi。2019。Diskann：单节点上的快速、精确十亿点规模最近邻搜索。神经信息处理系统进展 32 (2019)。[33] Hervé Jégou、Matthijs Douze 和 Cordelia Schmid。2011。用于最近邻搜索的乘积量化。IEEE模式分析与机器智能汇刊 (TPAMI)。33, 1 (2011), 117–128。[34] Chao Jin、Zili Zhang、Xuanlin Jiang、Fangyue Liu、Xin Liu、Xuanzhe Liu 和 Xin Jin。2024。RAGCache：面向检索增强生成的高效知识缓存。CoRR abs/2404.12457 (2024)。[35] Jeff Johnson、Matthijs Douze 和 Hervé Jégou。2019。利用GPU进行十亿规模相似性搜索。IEEE大数据汇刊 7, 3 (2019), 535–547。[36] Saim Khan、Somesh Singh、Harsha Vardhan Simhadri、Jyothi Vedurada 等。2024。BANG：使用单个GPU进行十亿规模近似最近邻搜索。arXiv预印本 arXiv:2401.11324 (2024)。[37] Donghee Lee、Jongmoo Choi、Jong-Hun Kim、Sam H Noh、Sang Lyul Min、Yookun Cho 和 Chong Sang Kim。2001。LRFU：涵盖最近最少使用和最不经常使用策略的一系列策略。IEEE计算机汇刊 50, 12 (2001), 1352–1361。[38] 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。2020。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。[39] Jie Li、Haifeng Liu、Chuanghua Gui、Jianyu Chen、Zhenyuan Ni、Ning Wang 和 Yuan Chen。2018。京东电商平台实时视觉搜索系统的设计与实现。载于中间件国际会议论文集。9–16。[40] Ting Liu、Andrew Moore、Ke Yang 和 Alexander Gray。2004。实用近似最近邻算法研究。NIPS 17 (2004)。[41] Ruiyao Ma、Yifan Zhu、Baihua Zheng、Lu Chen、Congcong Ge 和 Yunjun Gao。2024。GTI：用于高维空间最近邻搜索、支持对数更新的基于图的树索引。VLDB基金会会刊 18, 4 (2024), 986–999。[42] Yury A. Malkov 和 Dmitry A. Yashunin。2020。使用分层可导航小世界图实现高效、稳健的近似最近邻搜索。IEEE模式分析与机器智能汇刊 (TPAMI)。42, 4 (2020), 824–836。[43] Magdalen Dobson Manohar、Zheqi Shen、Guy Blelloch、Laxman Dhulipala、Yan Gu、Harsha Vardhan Simhadri 和 Yihan Sun。2024。ParlayANN：可扩展且确定性的并行基于图的近似最近邻搜索算法。载于SIGPLAN并行编程原理与实践年度研讨会论文集。270–285。[44] Jason Mohoney、Devesh Sarda、Mengze Tang、Shihabur Rahman Chowdhury、Anil Pacaci、Ihab F. Ilyas、Theodoros Rekatsinas 和 Shivaram Venkataraman。2025。Quake：向量搜索的自适应索引。载于USENIX操作系统设计与实现会议论文集。文章 9，共 17 页。[45] NVIDIA。2025。向RAG流式传输数据。https://build.nvidia.com/nvidia/streaming-data-to-rag/blueprintcard。[46] Elizabeth J. O’Neil、Patrick E. O’Neil 和 Gerhard Weikum。1993。用于数据库磁盘缓冲的LRU-K页面置换算法。载于国际数据管理会议论文集 (SIGMOD)。297–306。[47] Hiroyuki Ootomo、Akira Naruse、Corey Nolet、Ray Wang、Tamas Feher 和 Yong Wang。2024。CAGRA：面向GPU的高度并行图构建与近似最近邻搜索。载于IEEE数据工程国际会议 (ICDE)。4236–4247。[48] Charles Packer、Vivian Fang、Shishir G. Patil、Kevin Lin、Sarah Wooders 和 Joseph E. Gonzalez。2023。MemGPT：迈向作为操作系统的大语言模型。CoRR abs/2310.08560 (2023)。[49] Y Peng、B Choi、TN Chan、J Yang 和 J Xu。2023。多维数据库中的高效近似最近邻搜索。国际数据管理会议论文集 (SIGMOD)。(2023)。[50] Yandex Research。2021。十亿规模相似性搜索基准。https: //research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search。[51] Chenghao Rong、Jessie Hui Wang、Juncai Liu、Jilong Wang、Fenghua Li 和 Xiaolei Huang。2021。调度大规模摄像头流以优化大规模实时视频分析。IEEE/ACM网络汇刊 30, 2 (2021), 867–880。[52] Harsha Vardhan Simhadri、Martin Aumüller、Amir Ingber、Matthijs Douze、George Williams、Magdalen Dobson Manohar、Dmitry Baranchuk、Edo Liberty、Frank Liu、Ben Landrum 等。2024。大规模近似最近邻竞赛结果：NeurIPS’23竞赛。arXiv预印本 arXiv:2409.17424 (2024)。

[53] Aditi Singh、Suhas Jayaram Subramanya、Ravishankar Krishnaswamy 和 Harsha Vardhan Simhadri。2021。FreshDiskANN：面向流式相似性搜索的快速、精确基于图的近似最近邻索引。*arXiv*预印本 *arXiv:2105.09613* (2021)。[54] Rafael Souza、André Fernandes、Thiago SFX Teixeira、George Teodoro 和 Renato Ferreira。2021。在CPU–GPU平台上通过自适应工作划分实现在线多媒体检索。*J. Parallel and Distrib. Comput.* 148 (2021), 31–45。[55] Michael Stonebraker 和 Andrew Pavlo。2024。周而复始……又周而复始。*SIGMOD Rec.* 53, 2 (2024), 21–37。[56] Yiping Sun、Yang Shi 和 Jiaolong Du。2024。用于在线近似最近邻搜索的实时自适应多流GPU系统。载于第*33*届*ACM*信息与知识管理国际会议论文集。4906–4913。[57] Bing Tian、Haikun Liu、Yuhang Tang、Shihai Xiao、Zhuohui Duan、Xiaofei Liao、Hai Jin、Xuecang Zhang、Junhua Zhu 和 Yu Zhang。2025。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*USENIX*文件与存储技术会议 *(FAST)*。171–185。[58] Yao Tian、Xi Zhao 和 Xiaofang Zhou。2023。DB-LSH 2.0：结合基于查询的动态分桶的局部敏感哈希。*IEEE*知识与数据工程汇刊 36, 3 (2023), 1000–1015。[59] Nitish Upreti、Krishnan Sundaram、Hari Sudan Sundar、Samer Boshra、Balachandar Perumalswamy、Shivam Atri、Martin Chisholm、Revti Raman Singh、Greg Yang、Subramanyam Pattipaka 等。2025。Azure Cosmos DB上的高性价比、低延迟向量搜索。*arXiv*预印本 *arXiv:2505.05885* (2025)。[60] Mengzhao Wang、Xiaoliang Xu、Qiang Yue 和 Yuxiang Wang。2021。基于图的近似最近邻搜索：全面综述与实验比较。*VLDB*基金会会刊 14, 11 (2021年7月), 1964–1978。[61] Wentao Xiao、Yueyang Zhan、Rui Xi、Mengshu Hou 和 Jianming Liao。2024。增强HNSW索引以支持实时更新：解决不可达点和性能下降问题。*arXiv*预印本 *arXiv:2407.07871* (2024)。[62] Haike Xu、Magdalen Dobson Manohar、Philip A. Bernstein、Badrish Chandramouli、Richard Wen 和 Harsha Vardhan Simhadri。2025。流式近似最近邻搜索中图索引的原地更新。*CoRR* abs/2502.13826 (2025)。

[63] 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。2023年。SPFresh：面向十亿级向量搜索的增量原地更新。发表于第*29*届操作系统原理研讨会（*SOSP*）论文集。545–561。[64] Shuo Yang、Jiadong Xie、Yingfan Liu、Jeffrey Xu Yu、Xiyue Gao、Qianru Wang、Yanguo Peng 和 Jiangtao Cui。2025年。重新审视基于邻近图的近似最近邻搜索的索引构建。*VLDB*基金会会刊 18，6（2025年），1825–1838。[65] Yuanhang Yu、Dong Wen、Ying Zhang、Lu Qin、Wenjie Zhang 和 Xuemin Lin。2022年。GPU加速的邻近图近似最近邻搜索与构建。发表于*2022*年*IEEE*第*38*届国际数据工程会议（*ICDE*）。IEEE，552–564。[66] Yuxiang Zeng、Yongxin Tong 和 Lei Chen。2023年。LiteHST：一种基于树嵌入的相似性搜索方法。*ACM*数据管理会刊 1，1（2023年），35:1–35:26。[67] Jianjin Zhang、Zheng Liu、Weihao Han、Shitao Xiao、Ruicheng Zheng、Yingxia Shao、Hao Sun、Hanqing Zhu、Premkumar Srinivasan、Weiwei Deng、Qi Zhang 和 Xing Xie。2022年。Uni-Retriever：迈向必应赞助商搜索中的统一嵌入式检索器学习。发表于*ACM*国际知识发现与数据挖掘会议（*KDD*）论文集。4493–4501。[68] Zili Zhang、Fangyue Liu、Gang Huang、Xuanzhe Liu 和 Xin Jin。2024年。通过重排序流水线，在超出GPU内存的大型数据集上实现快速向量查询处理。发表于*USENIX*网络系统设计与实现研讨会（*NSDI*）。23–40。[69] Weijie Zhao、Shulong Tan 和 Ping Li。2020年。SONG：GPU上的近似最近邻搜索。发表于*IEEE*数据工程国际会议（*ICDE*）。1033–1044。[70] Bolong Zheng、Zhao Xi、Lianggui Weng、Nguyen Quoc Viet Hung、Hang Liu 和 Christian S Jensen。2020年。PM-LSH：用于高维近似近邻搜索的快速、准确的LSH框架。*VLDB*基金会会刊 13，5（2020年），643–655。[71] Chaoji Zuo、Miao Qiao、Wenchao Zhou、Feifei Li 和 Dong Deng。2024年。SeRF：用于范围过滤近似最近邻搜索的分段图。*ACM*数据管理会刊（*SIGMOD*）2，1（2024年），1–26。