近似最近邻搜索（ANNS）

大模型视角下的近似最近邻搜索

陈力峥
lizhengchen@smail.nju.edu.cn

2025 年 9 月 19 日

Question

Question: 有哪些非结构化数据？以什么形式输入给计算机？

研究背景（1/8）

- 非结构化数据 $\xrightarrow{embedding}$ 向量 $\xrightarrow{input}$ 计算机
- 向量的相似度可刻画数据的语义相似度
- 因此相似性搜索 = 向量空间中的最近邻搜索 (NNS)

图: 非结构化数据应用

研究背景（2/8）

手写字体识别

- **MNIST 数据集**
- 28x28 784 dimension

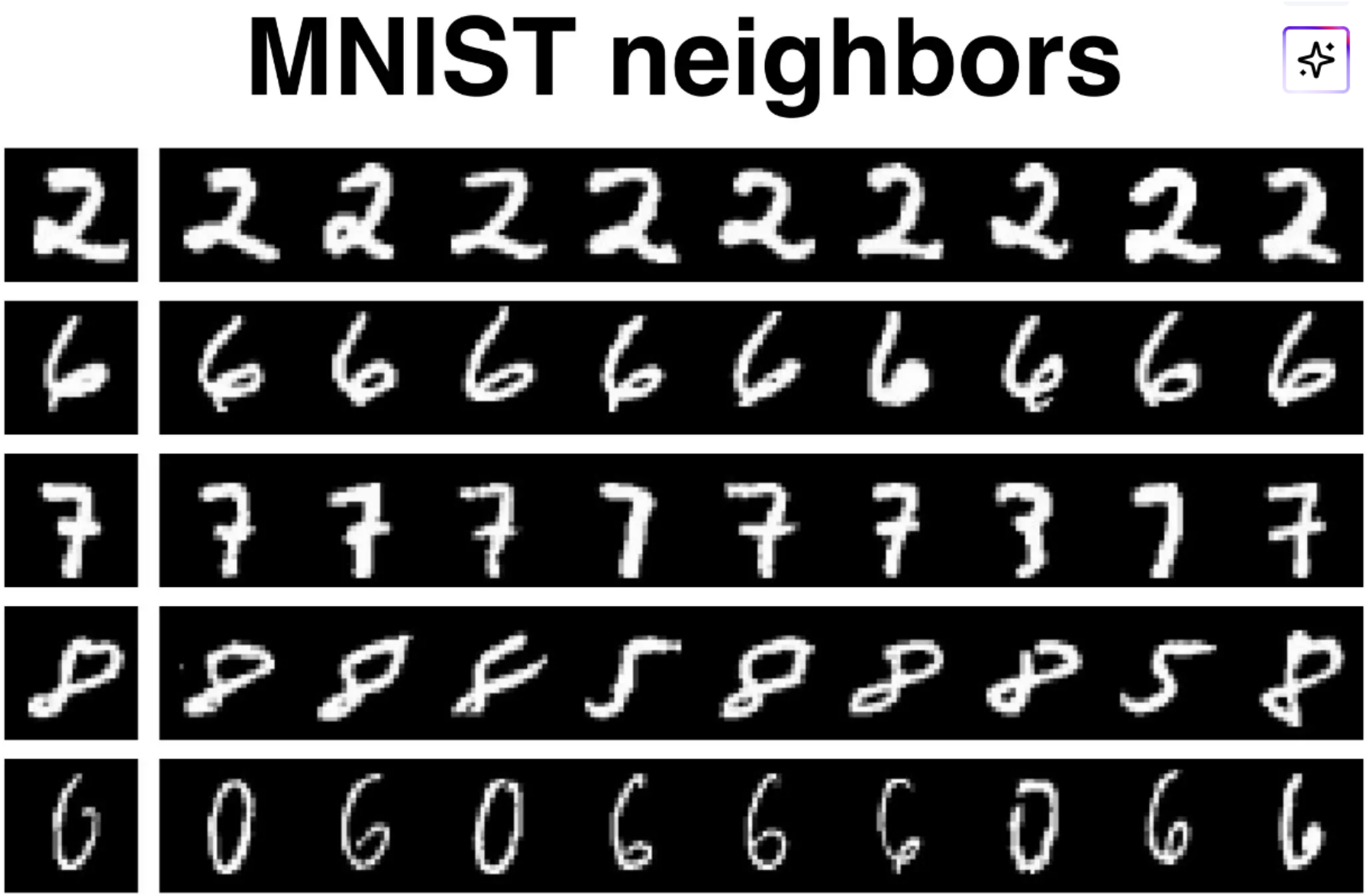
使用 L2 距离定义：

$$d(a,b) = \sum_{1 \le i,j \le 28} (a_{ij} - b_{ij})^2$$

$$d(\ \ ,\ \ ) = 53.6424865723$$

图：MNIST Distance

研究背景（3/8）



图：MNIST Neighbors

研究背景（4/8）

最近邻搜索 (Nearest Neighbor Search, NNS)

定义

给定一个向量集合 $\mathcal{X} = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$ 和查询向量 $q \in \mathbb{R}^d$，**最近邻搜索的目标是找到**

$$x^* = \arg\min_{x \in \mathcal{X}} \text{dist}(q, x),$$

其中 $\text{dist}(\cdot, \cdot)$ 为距离度量。

Question

Question: 有哪些常用的距离/相似度度量方式？

研究背景（5/8）

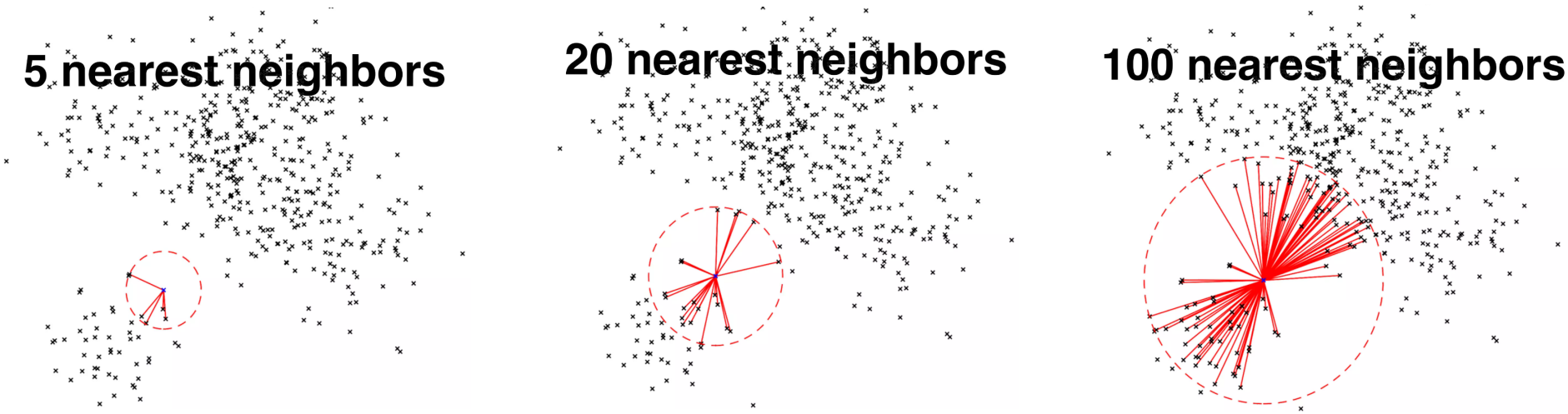


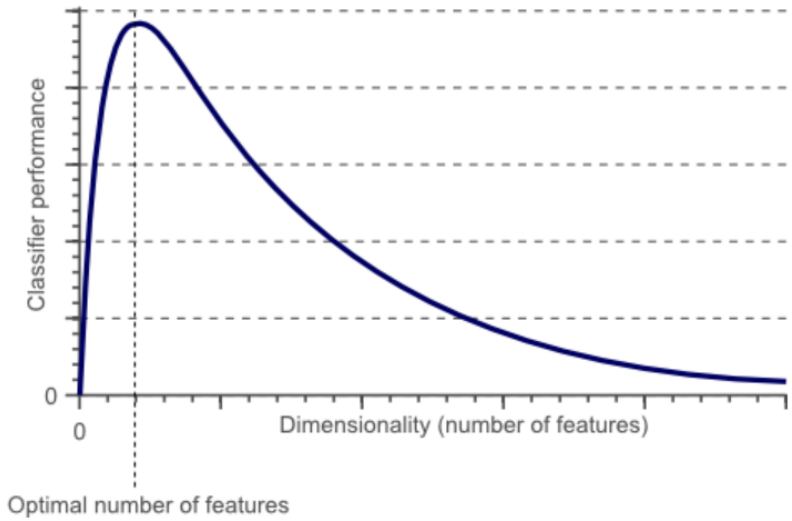
图: eg. Nearest Neighbor Search

研究背景（6/8）

精确最近邻搜索的挑战

- 数据规模从 **百万级**扩展到 **十亿级**
- 向量维度达到 **数百** ∼ **上千**
- 计算代价与存储开销 **急剧增加**

引出：需要在牺牲少量精度的情况下，换取大幅提升的效率 ⇒ **近似最近邻搜索 (ANNS)**



图：维度灾难

研究背景（7/8）

近似最近邻搜索 (Approximate NNS, ANNS)

定义

给定集合 $\mathcal{X} = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$，查询向量 $q \in \mathbb{R}^d$：

- **(1+ϵ)-ANNS**：找到 x'，使得

$$\mathrm{dist}(q, x') \leq (1+\epsilon) \cdot \mathrm{dist}(q, x^*),$$

  其中 x^* 为真实最近邻，$\epsilon \geq 0$。
- **Recall@k**：在返回的 k 个结果中，包含真实 Top-k 的比例。

研究背景（8/8）

- 随着大语言模型（LLM）的发展，ANNS 已成为其外部知识接入与语义检索的关键技术。
- 在 RAG （检索增强生成）中：
  - 快速检索相关的知识向量表示
  - 用外部信息增强大语言模型

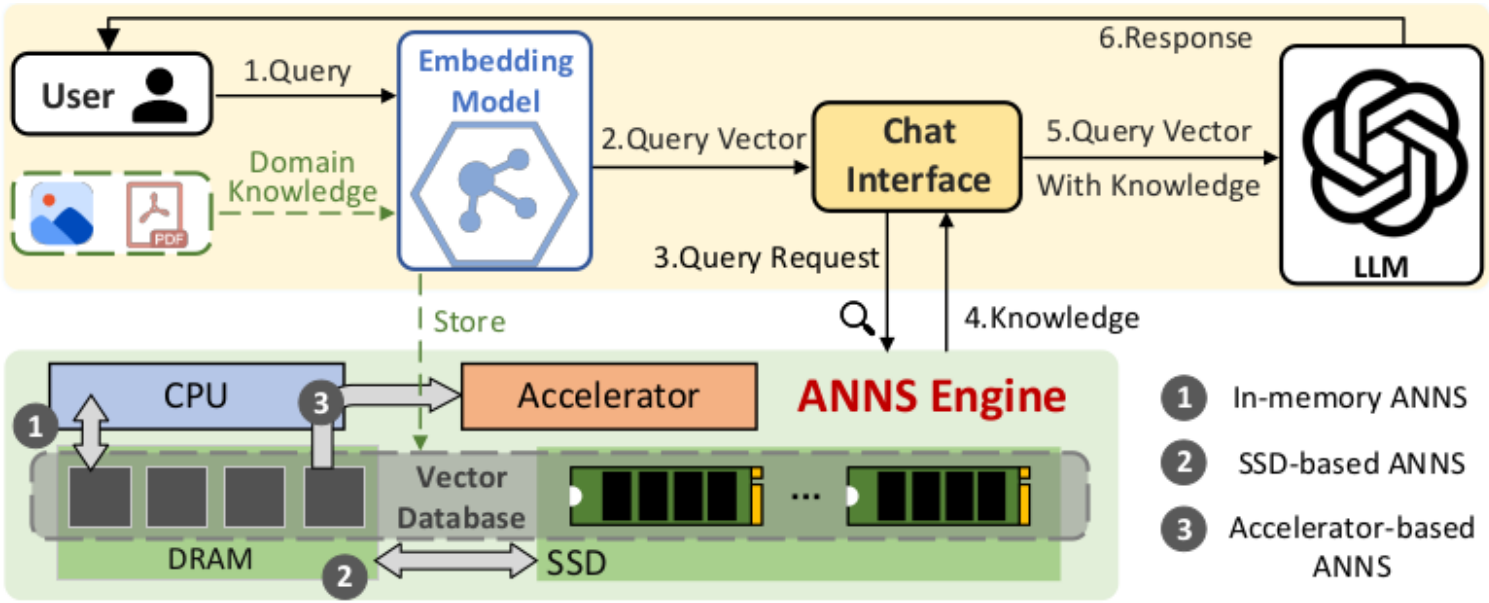


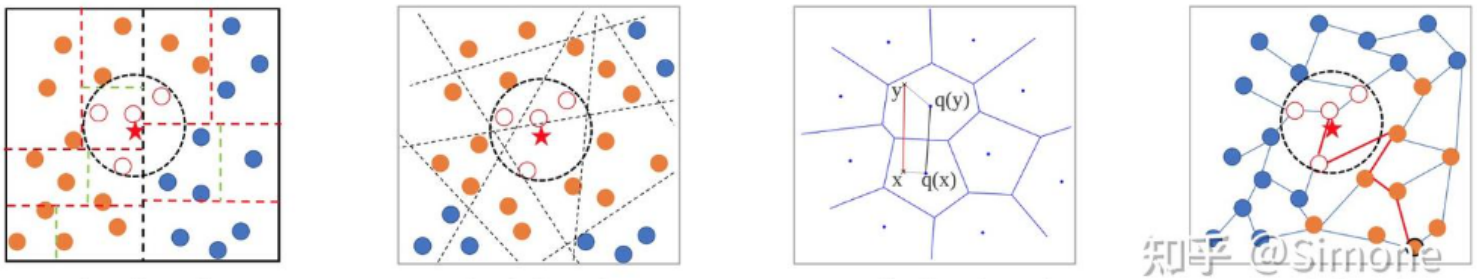
图: LLM With ANNS

ANNS 索引算法（1/12）

- 定义向量相似度后，如何快速找到相似数据？
- Naive Solution：遍历全量向量，计算查询 q 与每个向量的距离
- 在大规模高维数据中：
  1. 数据集规模巨大
  2. 维度灾难，向量难以区分
  3. 精确最近邻搜索代价高昂
- 需要高效的索引结构来支持近似最近邻搜索 (ANNS)

ANNS 索引算法（2/12）

- 主流的 ANNS 索引策略：
  1. Tree-based
  2. Hashing-based
  3. Quantization-based
  4. Graph-based
- 近年来，研究热点集中在 **Graph-based 方法**

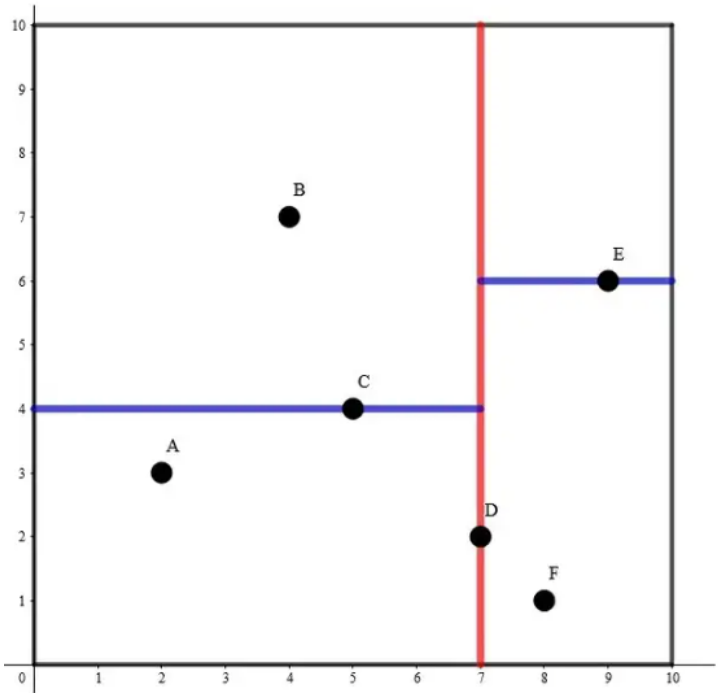


tree-based　hash-based　quantization-based　graph-based

图：主流 ANNS 索引策略示例

ANNS 索引算法（3/12）

- 划分 / 搜索
- 但对于高维数据表现不好 - Why？



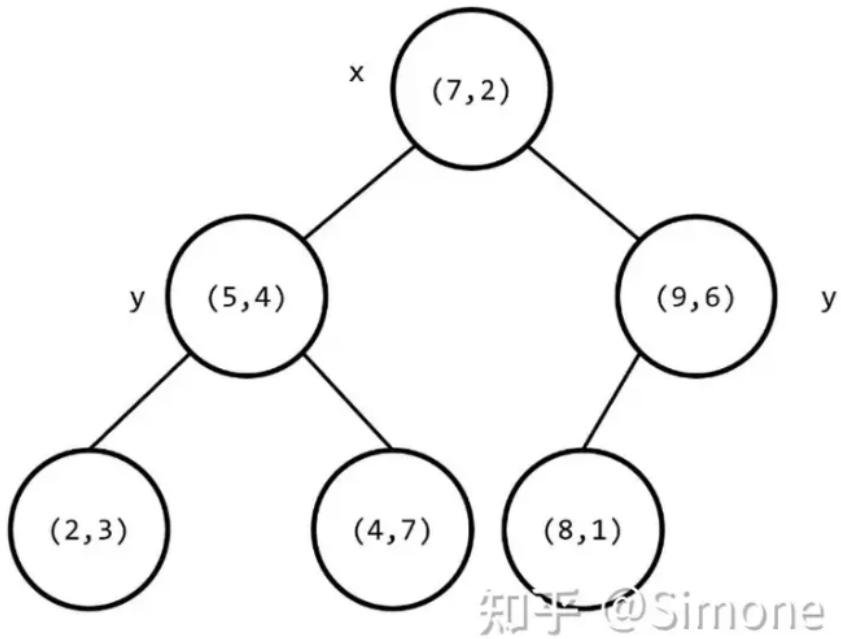


图: Tree-Based Solution: KD-Tree

ANNS 索引算法（4/12）

- **Question: Collision or Not?**
- 只需要在对应 bucket 中搜索

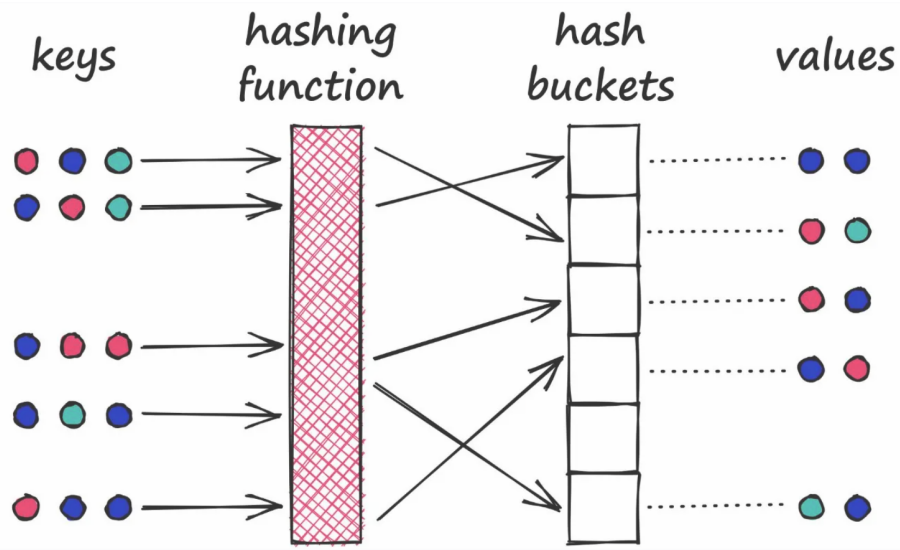


图: Normal Hash

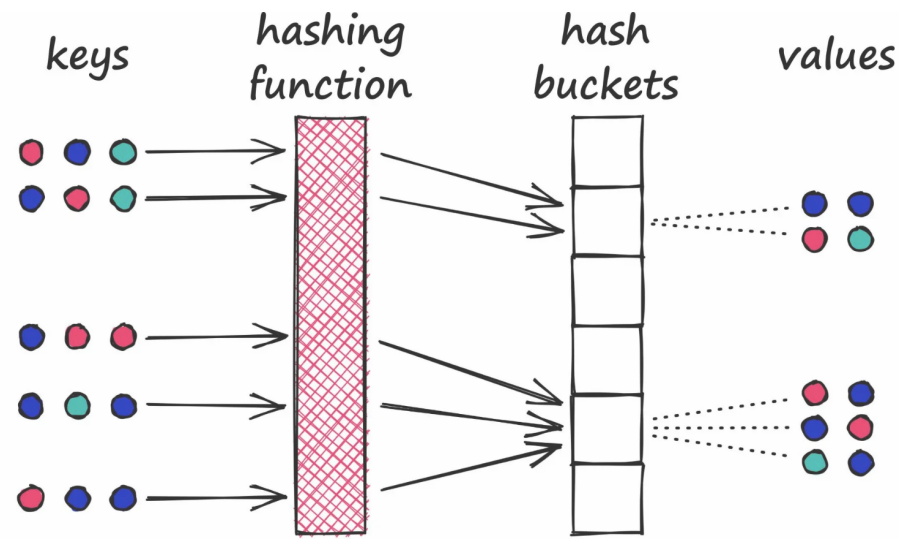
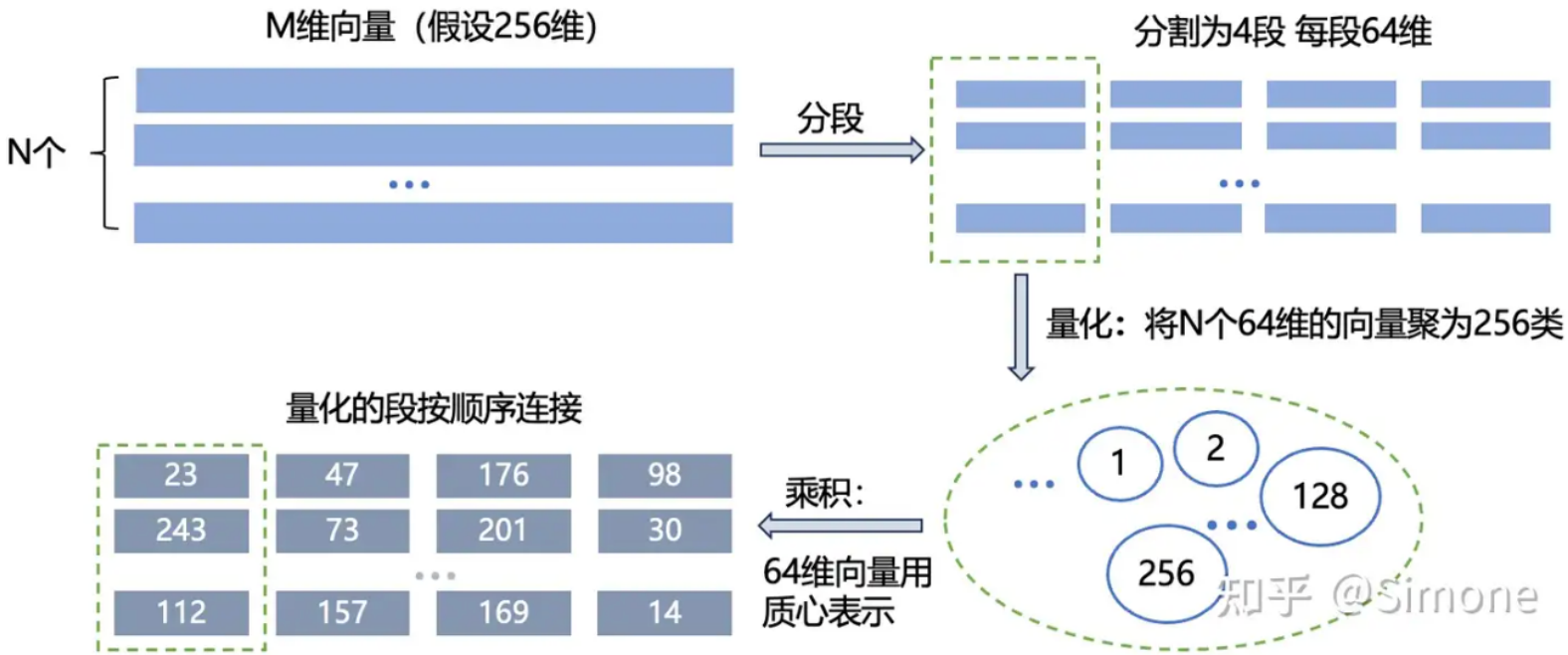


图: Locality Sensitive Hash

ANNS 索引算法（5/12）

- 向量分割：将高维向量分割为 B 个子向量，每个子向量维度为 D/B
- 量化：对子向量进行量化，将其映射到有限的码本上（使用聚类算法如 K-means）
- 编码：每个子向量通过一个索引表示，显著减少存储需求



图：Product Quantization

ANNS 索引算法（6/12）

- 图能很好地表达“邻居”的语义，在 ANNS 中展现了相比其他方法更优异的性能。
- 在构建的 APG 上进行贪心搜索，复杂度可达到 $\mathbf{O}(\log n)$

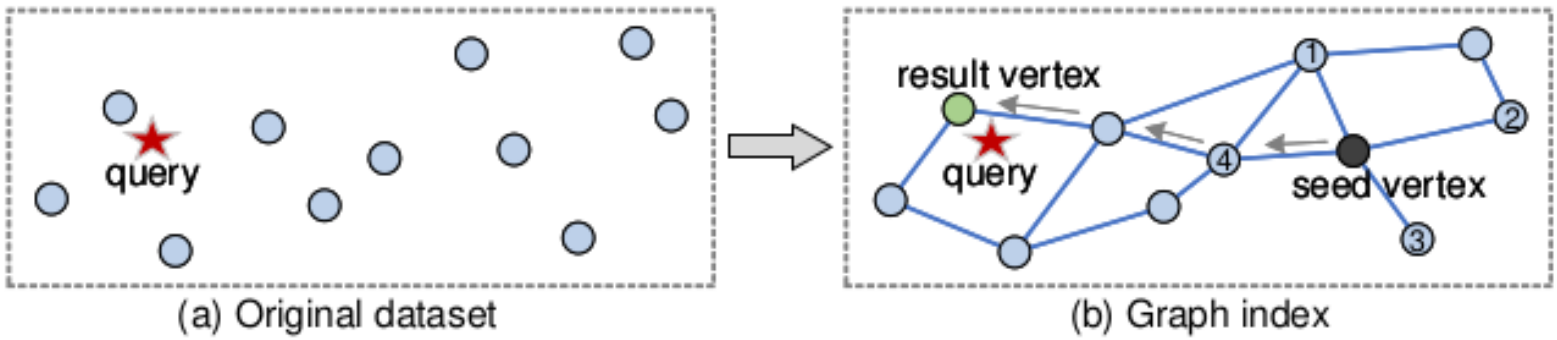


图: Graph Index.

ANNS 索引算法（7/12）

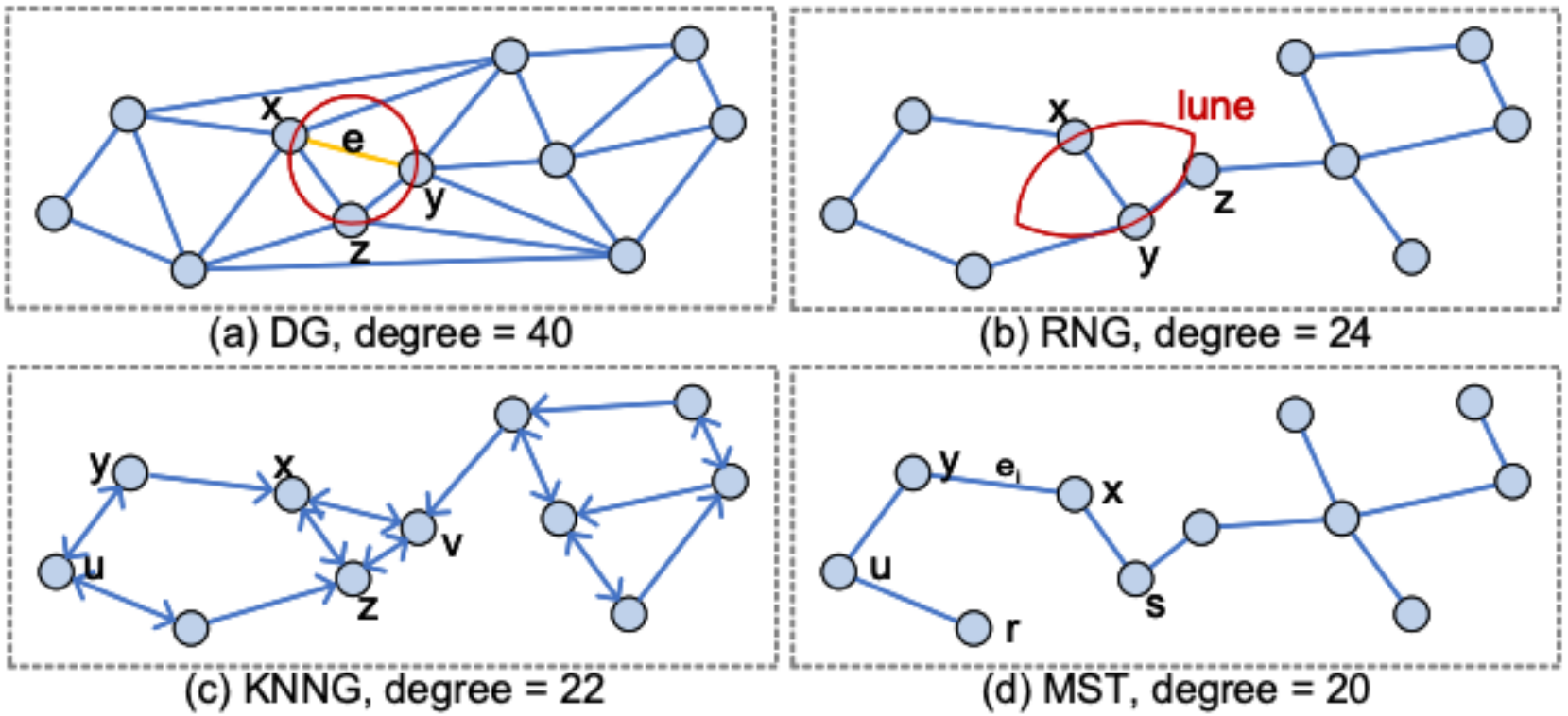
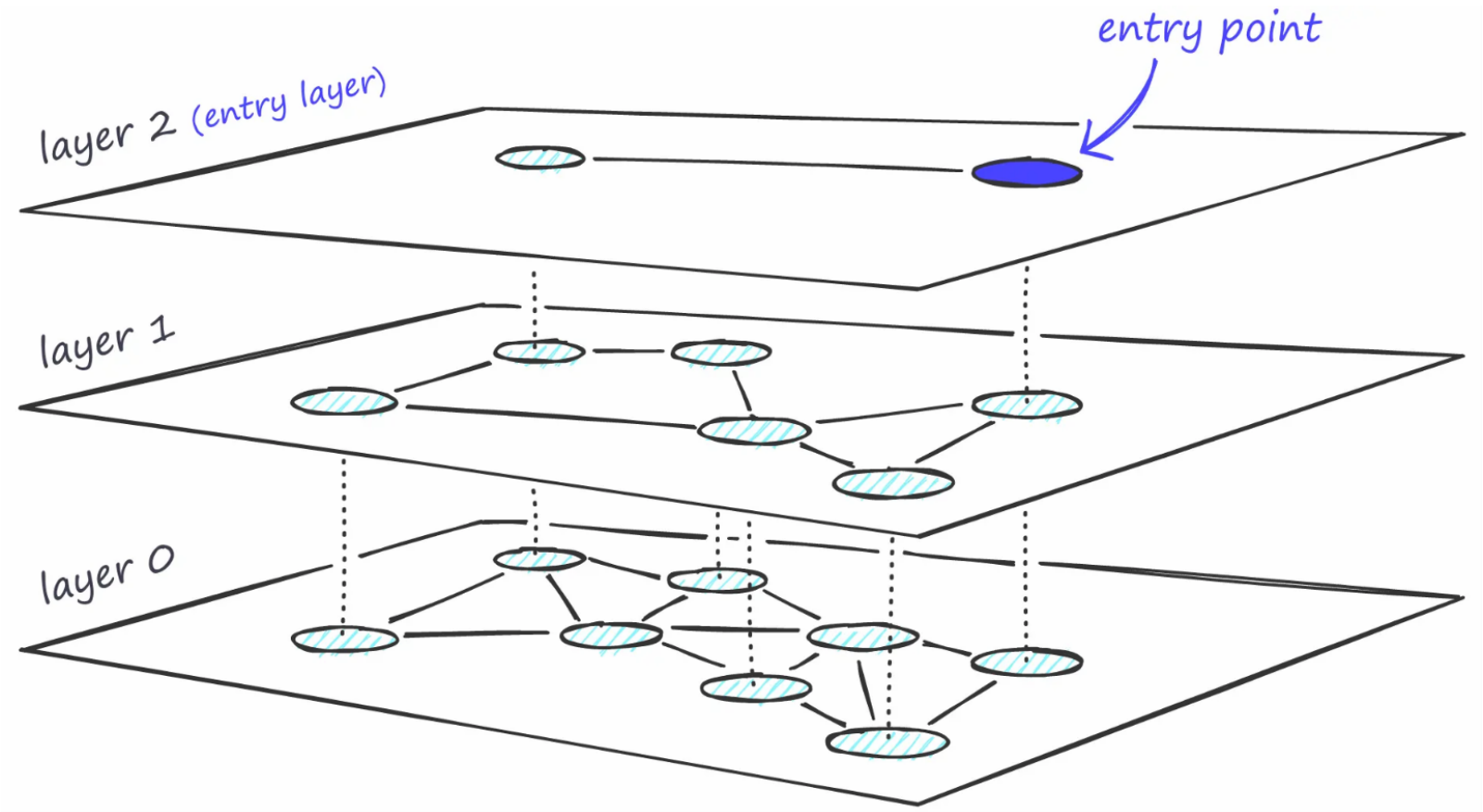


图: Approximate Proximity Graph.

ANNS 索引算法（8/12）

基于内存的图索引代表：HNSW

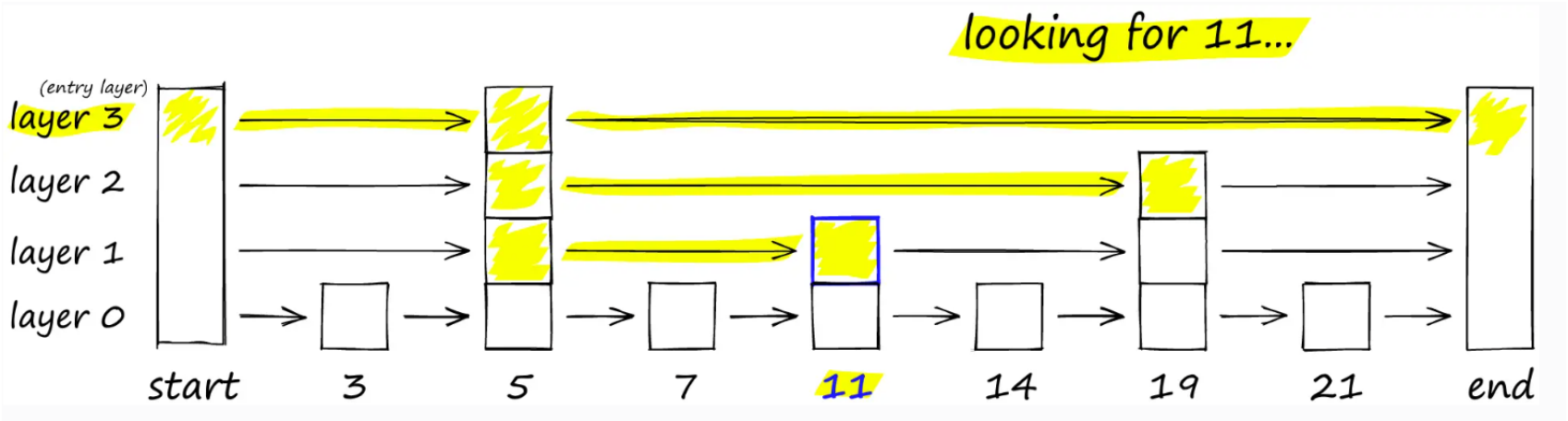


图：Hierarchical Navigable Small Worlds.

Malkov & Yashunin, TPAMI 2018. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World

ANNS 索引算法（9/12）

类比 SkipList 的搜索过程：



图：Skip List.

ANNS 索引算法（10/12）

基于内存的图索引代表：HNSW

Quesion: 从 SkipList 类似的搜索思路来看 HNSW 的搜索方式？

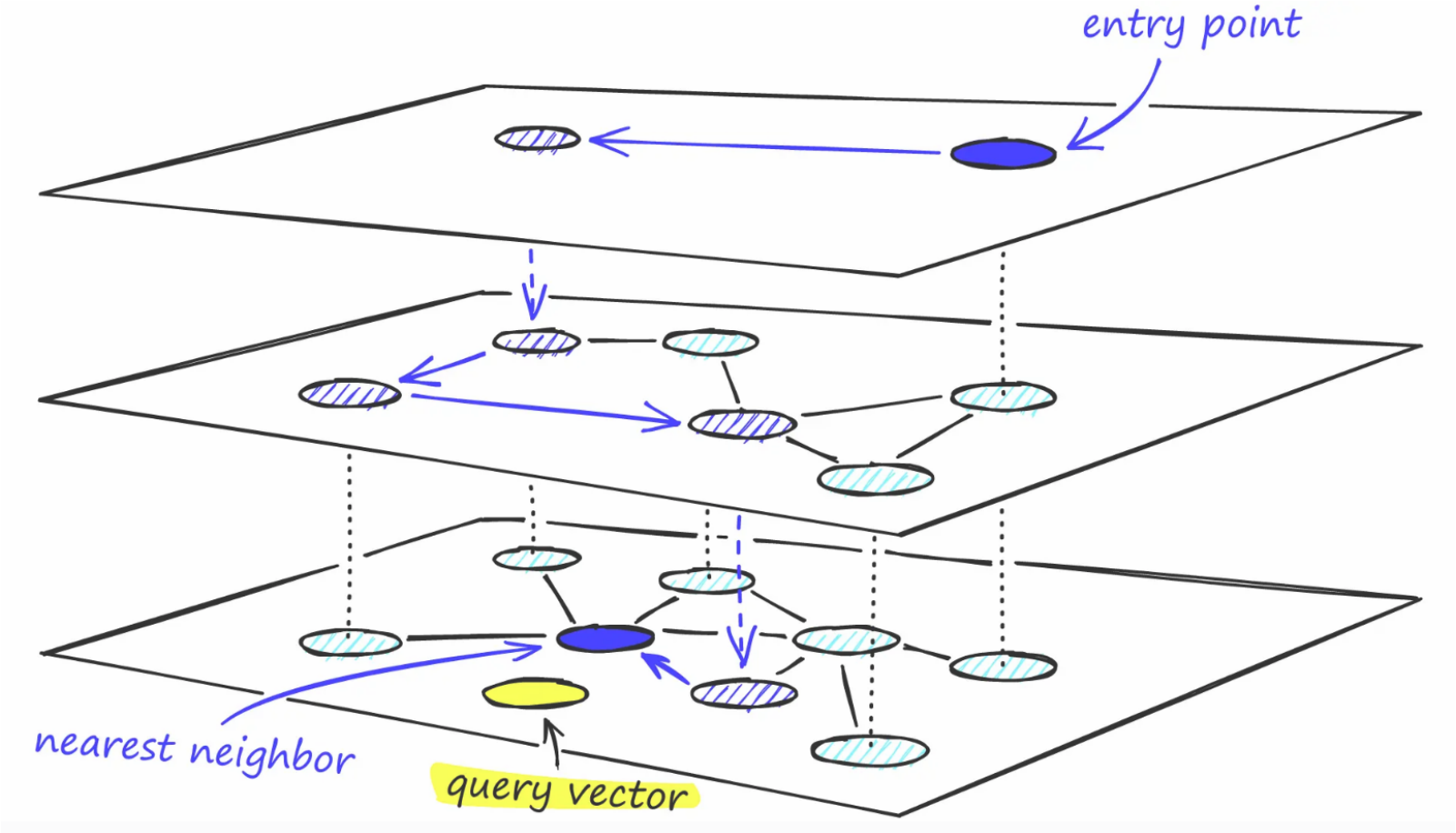


图: Search on Hierarchical Navigable Small Worlds.

ANNS 索引算法（11/12）

基于**磁盘**的图索引代表：**DiskANN** (Microsoft)
主要特点：

- 提供 Vamana 图索引（下一页）
- 支持在内存受限情况下构建图索引
- 支持在内存受限情况下对更大数据集进行检索
- 使用乘积量化（PQ）进一步减少内存占用

S. Jayaram Subramanya, F. Devvrit, et al., NeurIPS 2019. DiskANN: Fast Accurate Billion-Point Nearest Neighbor Search on a Single Node.

ANNS 索引算法（12/12）

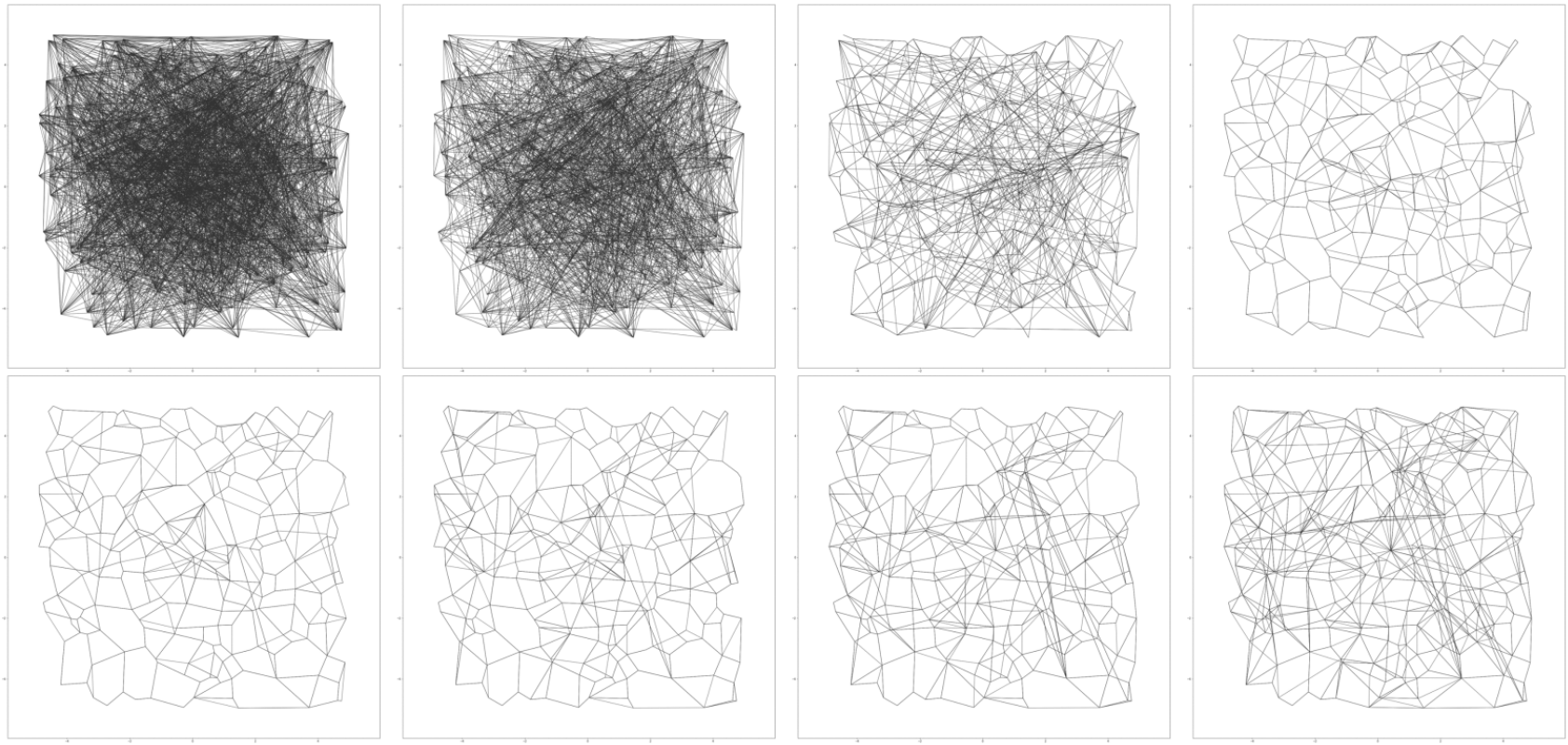
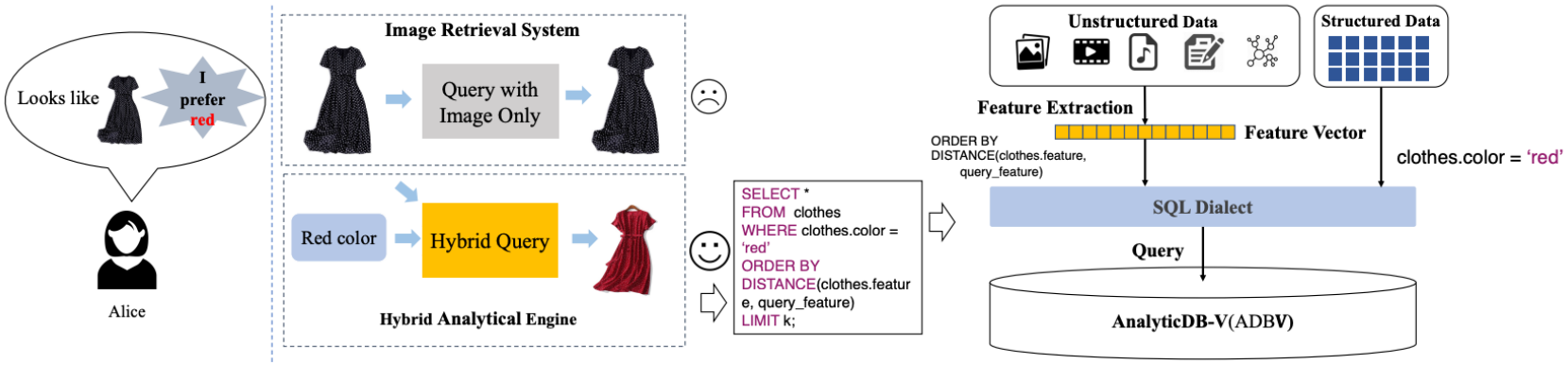


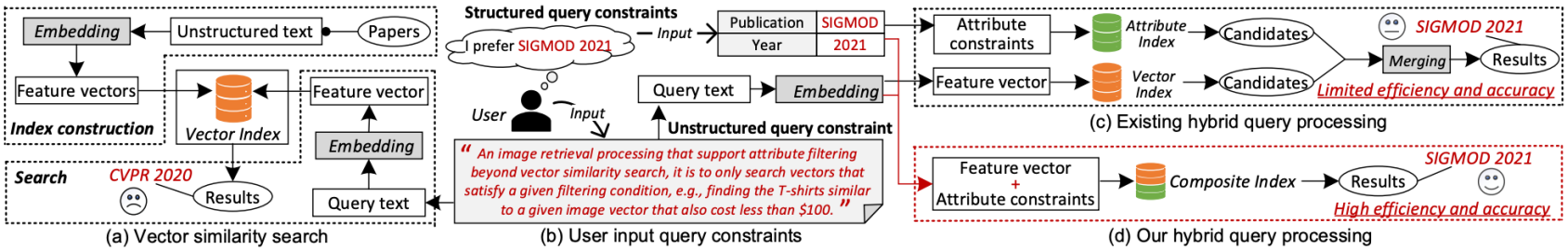
图: Vamana 图索引构建过程

S. Jayaram Subramanya, F. Devvrit, et al., NeurIPS 2019. DiskANN: Fast Accurate Billion-Point Nearest Neighbor Search on a Single Node.

研究方向概览（1/4）：ANNS 混合检索



图：Hybrid Query 1.



图：Hybrid Query 2.

研究方向概览（2/4）：ANNS 混合检索

单一模态或单一索引难以满足真实需求

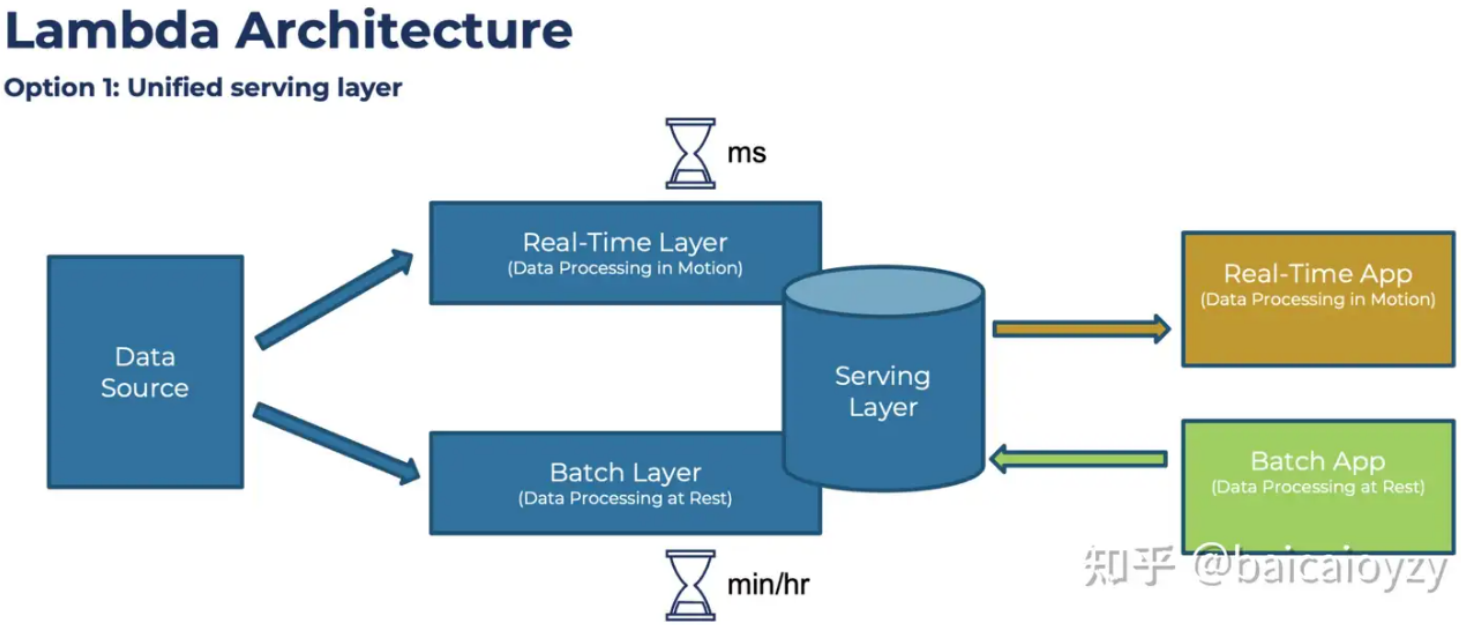
- 用户需求：多维条件结合 例如：”找评分最高的、同时和某主题最相关的文档”
- RAG 场景：**非结构化知识 + 数据库属性查询 + 知识图谱约束**
- 多模态检索：文本、图像、视频、音频等异构数据的统一查询

研究重点：

- 向量索引与传统数据库索引的高效协同
- 结构化与非结构化数据的统一检索

研究方向概览（3/4）：流式 ANNS

Stream/Dynamic 意味着需要 实时支持数据的增删改查。



图：大数据处理框架：Lambda.

研究方向概览（4/4）：流式 ANNS

传统 ANNS 系统的挑战：

- 传统 ANNS 系统多为 **静态索引**，构建后仅适合批量查询
- 难以应对数据的频繁更新，尤其是 **实时流数据**

在真实业务场景中：

- RAG 服务：知识库需实时扩充与更新，避免 LLM 检索滞后于最新事实
- 在线推荐与风控：用户行为与风险特征不断变化，要求索引支持快速插入与删除

流式 ANNS 的核心挑战：

- 保证 **低延迟查询**与 **高吞吐更新**
- 推动 **动态图索引结构**与增量维护策略的发展

AnalyticDB-V（1/7）

’现有’工作的缺陷 (Before AnalyticDB-V - Alibaba)

- **需要分别维护向量数据库 + 传统关系型数据库**
- **需要额外保证一致性**
- **难以联合优化，且分别查两个数据库延迟较高**
- **对于查询结果需要合并取交集，导致召回的数量不如预期（|Return| < k)**

W. Wei, et al., VLDB 2020. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data.

AnalyticDB-V（2/7）

- 关注其中流式 ANNS / 混合检索的设计

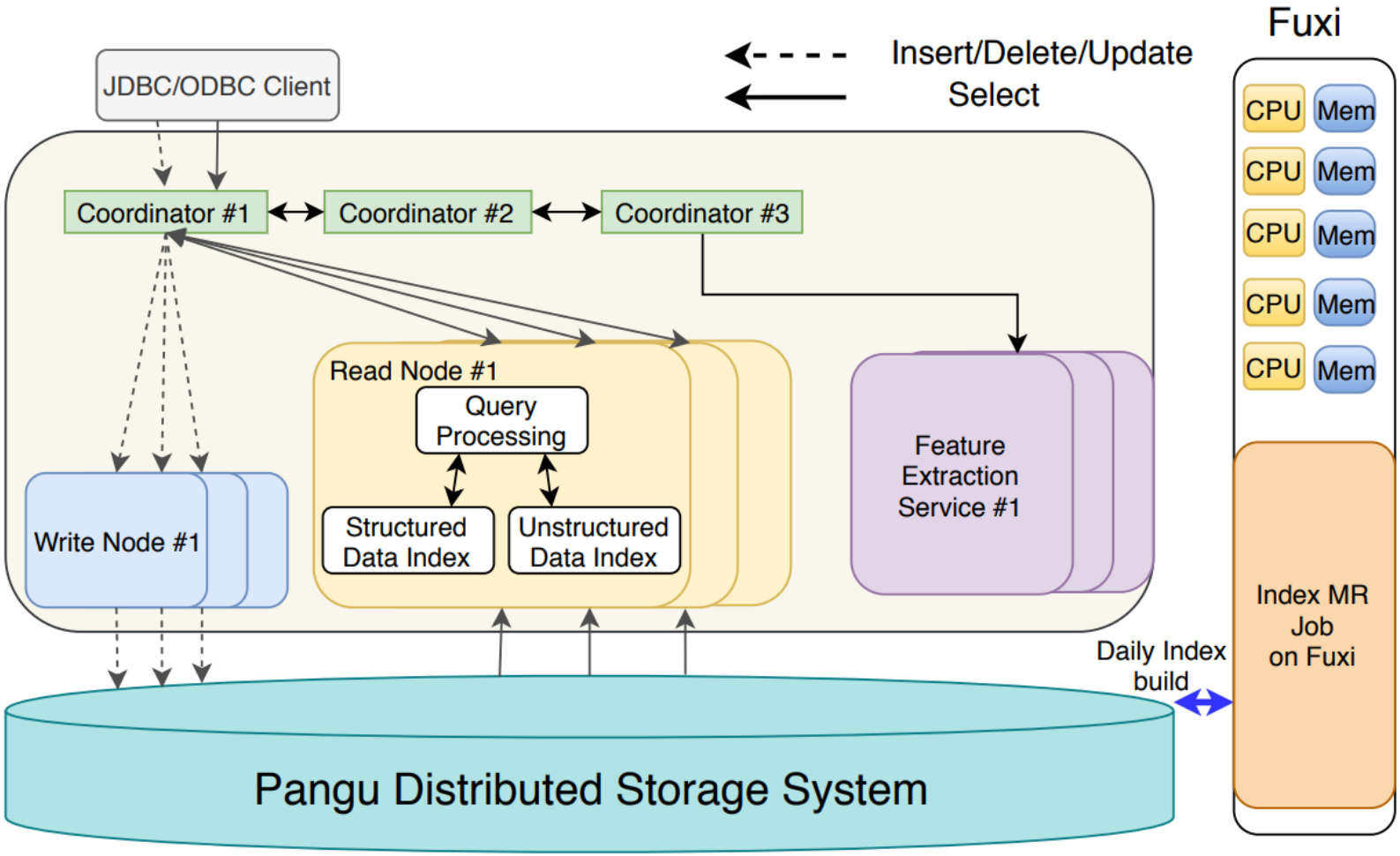


图: AnalyticDB-V Architecture.

AnalyticDB-V（3/7）

- 分为批处理/流处理两部分，兼顾时效性与资源开销

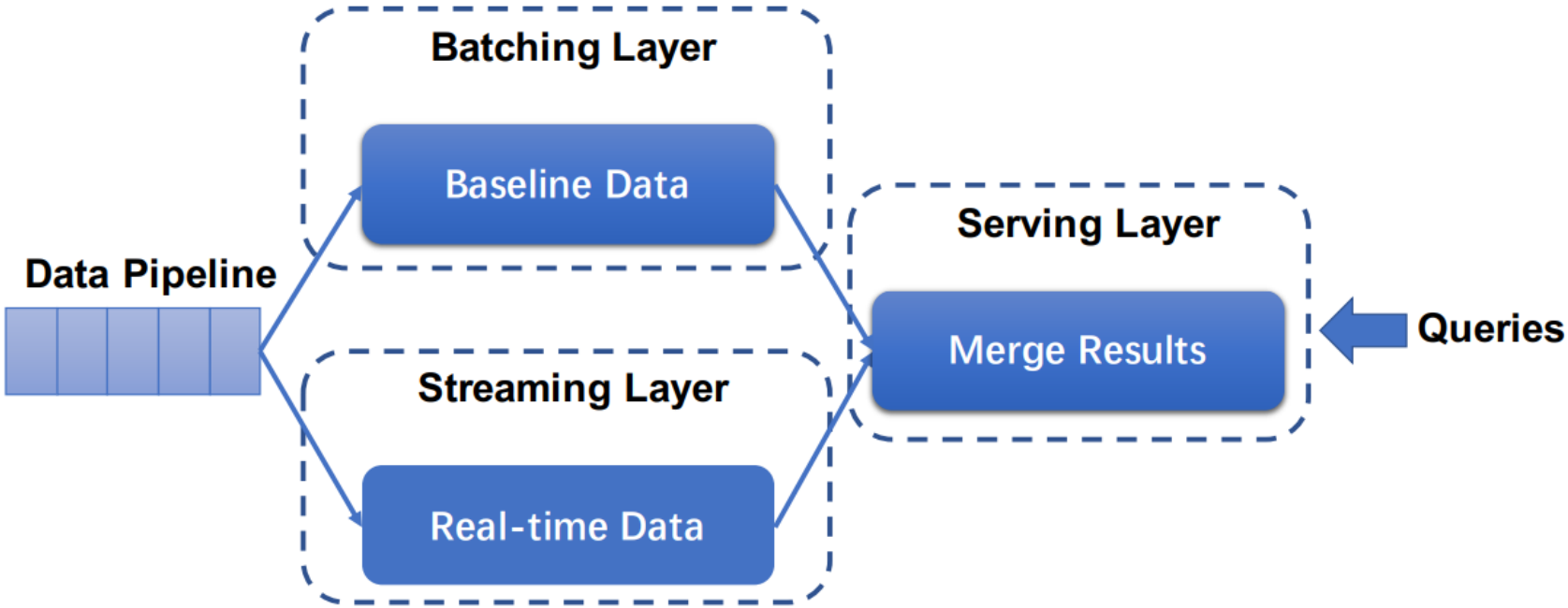


图: Lambda framework in AnalyticDB-V.

AnalyticDB-V（4/7）

- 合并 Baseline 数据与 Incremental 数据

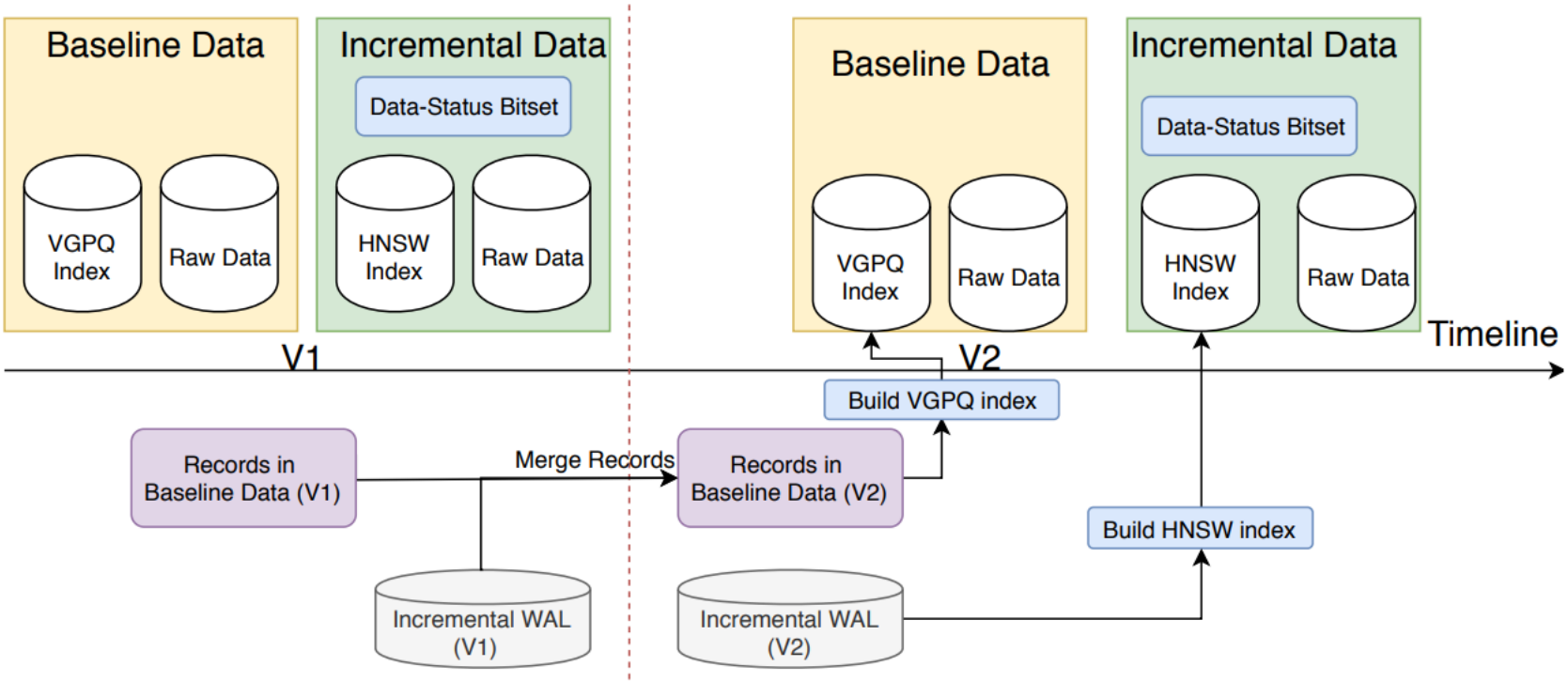
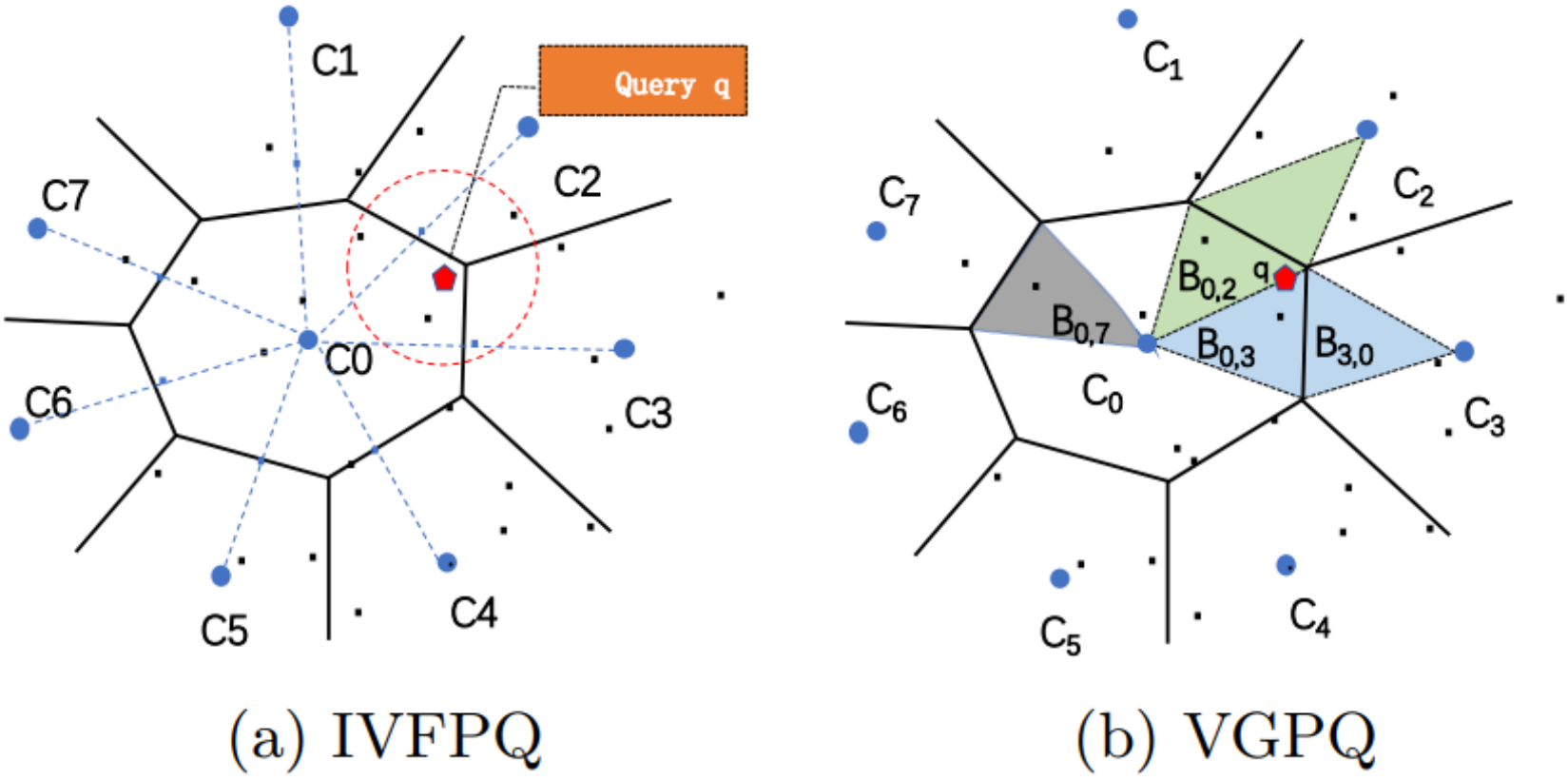


图: Merge Baseline Data And Incremental Data

Question: 哪种划分方式的搜索空间更大？



(a) IVFPQ (b) VGPQ

图: Voronoi Graph Product Quantization.

AnalyticDB-V（6/7）

混合检索策略：总体分为 **Pre-Filter** 与 **Post-Filter** 两大类。

1. Plan A: 先通过谓词过滤，再 Naive 地遍历所有符合过滤条件的向量.
2. Plan B: 先通过谓词过滤，运用 PQ 压缩的向量（预计算）来查找，但需要增大召回量，因为 PQ 会降低精度.
3. Plan C: 先通过谓词过滤，随后通过 VGPQ 图来查找近邻向量（其中利用过滤条件来剪枝）.
4. Plan D: 当满足谓词的数据高达百万量级时，需要后过滤策略来优化性能.

Question

Question: 分析 Pre-Filter 和 Post-Filter 两者的优势和劣势.

AnalyticDB-V（7/7）

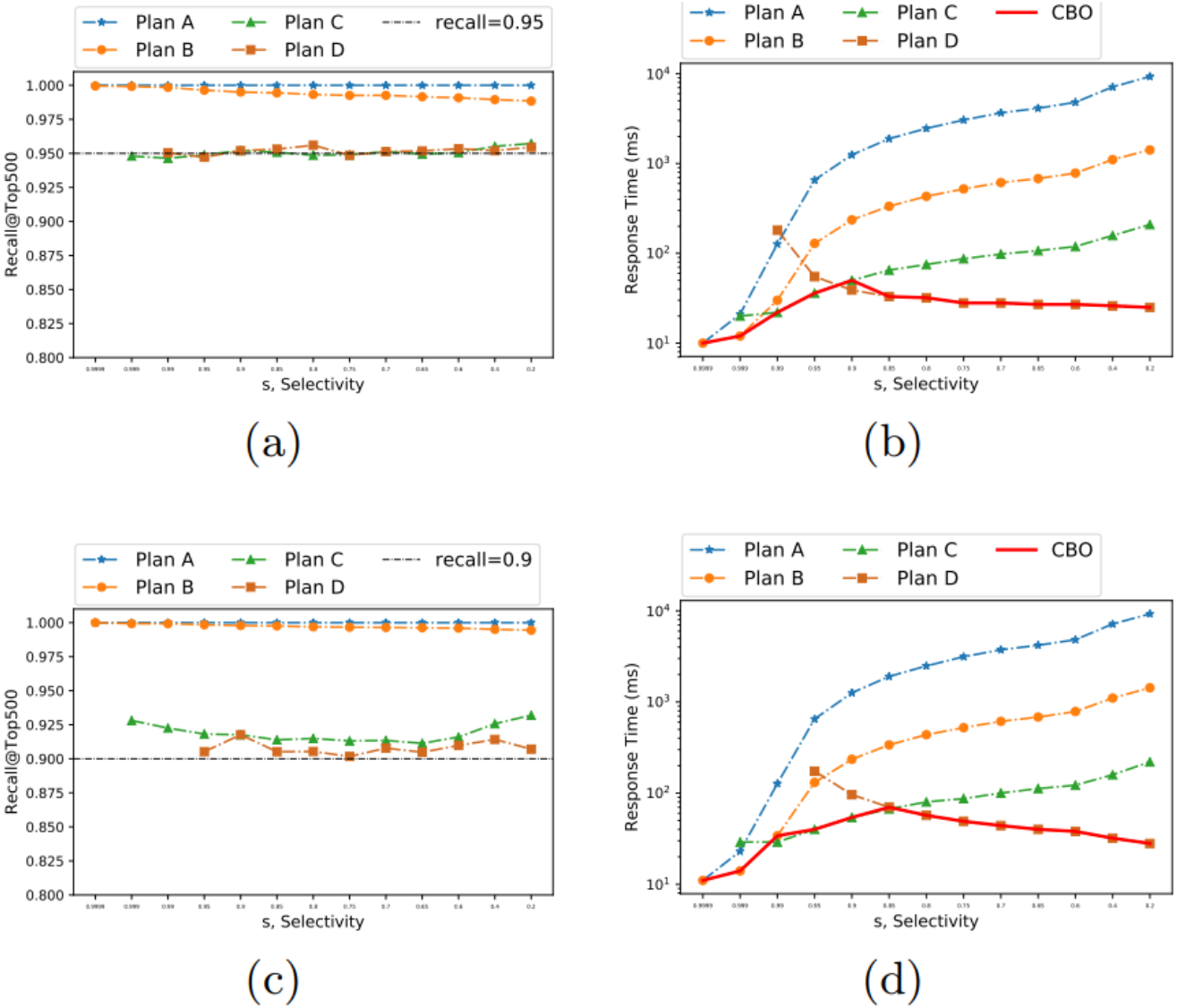


图: 对应 k = 50/250, 不同 Plan Recall/Latency 的表现.

FreshDiskANN（1/6）

现有算法的缺陷：

- **LSH (Local Sensitive Hashing)** 很容易支持流数据，但存在以下问题：
  - 需要极大的内存：存储额外的数据和 **上百个哈希函数**。
  - 查询速度慢：当索引存储在 **secondary storage** 时，查询非常慢。
- 现有工作：**PLSH (Parallel LSH)** 为分布式架构，在评估中：
  - 25 台性能相当的机器运行 PLSH，几乎只能与一台性能相当的 FreshDiskANN 相媲美（基于图的结构）。

A. Singh, S. J. Subramanya, R. Krishnaswamy, et al., arXiv 2021. FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search.

FreshDiskANN（2/6）

现有算法的缺陷：

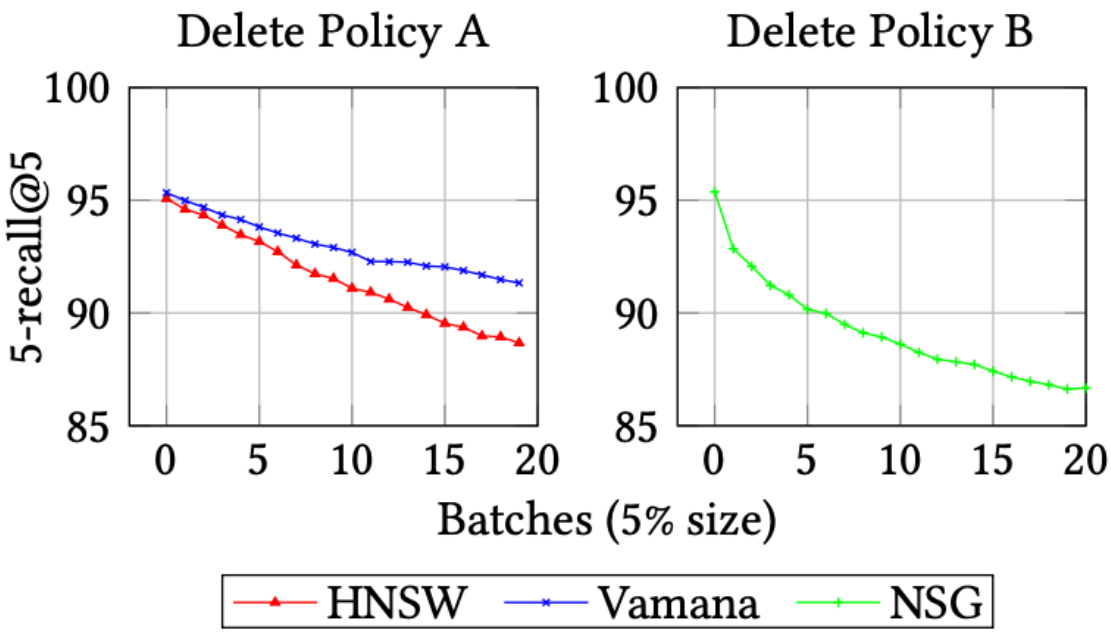
- 基于树的实现：
  - **HD-Index**：内存占用极低，但 **延迟很高**，召回率仅为 30%。
- **KD-Tree**：虽然支持快速更新，但随着维度增加，**检索效率大幅降低**。

引入图结构的原因：

- 图结构有更高的查询精度和速度，但对于插入和删除操作非常困难。
- 大多数现有系统采用从头 **重建索引**的方式，存在巨大的开销：
  - 对于 **1 亿点的 HNSW**，使用 **48 核心高端机器**需要约 2 小时重建。
  - 对于 **10 亿点的 HNSW**，需要使用 **三台机器**，才能保持每 6 小时进行一次重建更新。

插入与删除策略

- **Insertion Policy**：对于新点 p 的插入，运行候选生成算法，选择合适的出入边。
- **Delete Policy A**：删除点 p 时，直接移除与 p 相关的所有内外边，不补充任何新的边。
- **Delete Policy B**：删除点 p 时，移除其所有内外边，并在 p 的局部邻域中添加边：对于图中的任意一对有向边 (p_{in}, p) 和 (p, p_{out})，在更新图中添加边 $(p_{\text{in}}, p_{\text{out}})$。



- 基于 $\alpha - RNG$ 图的 Insert/Delete 策略

Algorithm 2: Insert(x_p, s, L, α, R)

Data: Graph $G(P, E)$ with start node s, new point to be added with vector x_p, distance threshold $\alpha > 1$, out degree bound R, search list size L
Result: Graph $G'(P', E')$ where $P' = P \cup \{p\}$
begin
  initialize set of expanded nodes $\mathcal{V} \leftarrow \emptyset$
  initialize candidate list $\mathcal{L} \leftarrow \emptyset$
  $[\mathcal{L}, \mathcal{V}] \leftarrow$ GreedySearch$(s, p, 1, L)$
  set p's out-neighbors to be $N_{out}(p) \leftarrow$ RobustPrune$(p, \mathcal{V}, \alpha, R)$ (Algorithm 3)
  foreach $j \in N_{out}(p)$ **do**
    if $|N_{out}(j) \cup \{p\}| > R$ **then**
      $N_{out}(j) \leftarrow$ RobustPrune$(j, N_{out}(j) \cup \{p\}, \alpha, R)$
    else
      update $N_{out}(j) \leftarrow N_{out}(j) \cup \{p\}$

Algorithm 4: Delete(L_D, R, α)

Data: Graph $G(P, E)$ with $|P| = n$, set of points to be deleted L_D
Result: Graph on nodes P' where $P' = P \setminus L_D$
begin
  foreach $p \in P \setminus L_D$ *s.t.* $N_{out}(p) \cap L_D \neq \emptyset$ **do**
    $\mathcal{D} \leftarrow N_{out}(p) \cap L_D$
    $C \leftarrow N_{out}(p) \setminus \mathcal{D}$ //initialize candidate list
    foreach $v \in \mathcal{D}$ **do**
      $C \leftarrow C \cup N_{out}(v)$
    $C \leftarrow C \setminus \mathcal{D}$
    $N_{out}(p) \leftarrow$ RobustPrune(p, C, α, R)

FreshDiskANN（5/6）

StreamingMerge 策略：持久化在 SSD 的 Long-Term Index (LTI) + 内存 RW-TempIndex。
当 TempIndex 超内存阈值 ⇒ 转为 RO-TempIndex 并触发后台合并。

1. **Delete Phase.**
   逐块加载 LTI，并行执行 `Delete` 去掉待删点及其边；
   必须扫描全部 LTI 块。
2. **Insert Phase.**
   对每棵新点 $p \in N$ 只在 SSD 图上做局部 `GreedySearch`；
   反向边先缓存在内存 Δ 结构，
   无需触碰全量 LTI。
3. **Patch Phase.**
   再次逐块读取 LTI，把 Δ 合并到各点出边并可能重新剪枝；
   必须再次扫描全部 LTI 块。

FreshDiskANN（6/6）

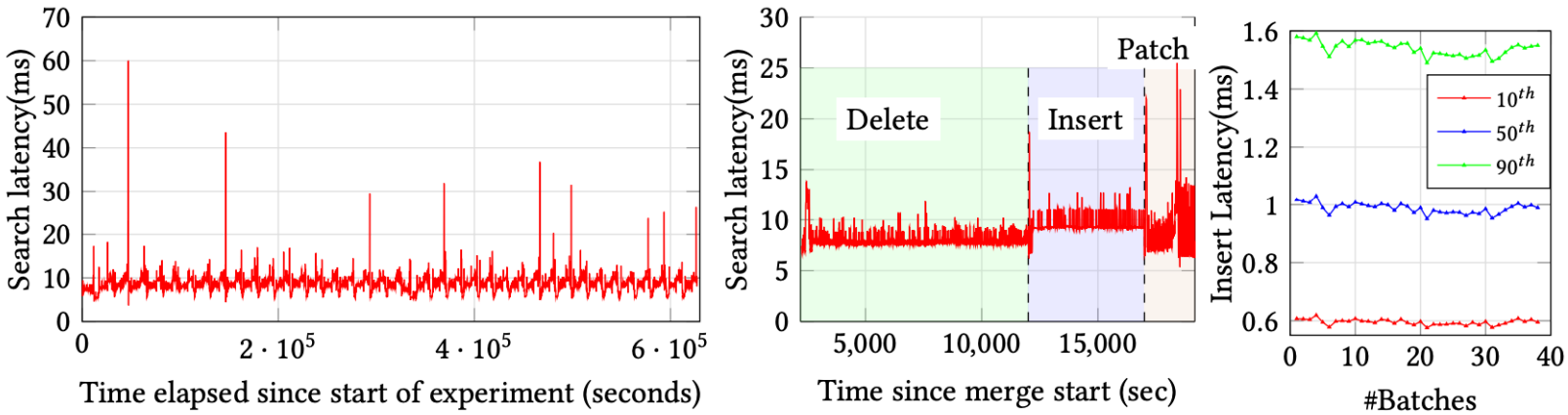


Figure 6. Mean latency[4]measurements for the week-long *steady-state* experiment with an 800M FreshDiskANN index processing concurrent inserts, deletes, and periodic background merge. (left) Search latency with $Ls = 100$ over the entire experiment; (middle) Search latency during one `StreamingMerge` run, zoomed in from the left plot; (right) 10^{th}, 50^{th} and 90^{th} percentile insert latency over the entire experiment.

Filtered-DiskANN（1/4）

- 现有大多数方法只修改 **搜索步骤**，而非索引构建步骤
- 三类常见方案：
  - **Post-processing**：先全量检索，再过滤 缺陷：低特异性过滤条件下需返回大量候选，效率低
  - **Pre-processing**：为每个过滤条件单独建索引 缺陷：过滤条件多或标签多时存储开销巨大
  - **Inline-processing**：在 IVF/LSH 索引中附带过滤信息 缺陷：底层索引效率有限

A. Gollapudi, et al., WWW 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters.

Filtered-DiskANN（2/4）

- Graph-based 索引（HNSW, Vamana 等）相比 IVF/LSH：
  - 查询代价更低，能在更少点比较下达到更高召回
  - 性能优势随数据规模增大而更显著
- 交互式服务（搜索、推荐、广告）几乎都依赖 Graph-based ANNS 实现毫秒级响应
- 现有 Filtered ANNS 方法没有充分利用 Graph-based 索引的效率优势，仅停留在 IVF/LSH 等结构上的优化

Filtered-DiskANN：将向量数据与标签融合到同一图索引之中，极大提高了搜索效率并且更好维护。但目前仅支持单一标签检索。

Filtered-DiskANN（3/4）

- 具体地，提出 Filtered-Vamana Graph.

Algorithm 1: FilteredGreedySearch(S, x_q, k, L, F_q)

Data: Graph G with initial nodes S, query vector x_q, search list size $\mathcal{L}$, and query filter(s) F_q.
Result: Result set $\mathcal{L}$ containing k approximate nearest neighbors, and a set $\mathcal{V}$ containing all visited nodes.
begin
1 Initialize sets $\mathcal{L} \leftarrow \emptyset$ and $\mathcal{V} \leftarrow \emptyset$.
for $s \in S$ **do**
if $F_s \cap F_x \neq \emptyset$ **then**
$\mathcal{L} \leftarrow \mathcal{L} \cup \{s\}$
while $\mathcal{L} \setminus \mathcal{V} \neq \emptyset$ **do**
2 Let $p^* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} \|x_p - x_q\|$
3 $\mathcal{V} \leftarrow \mathcal{V} \cup \{p^*\}$
4 Let $N'_{\text{out}}(p^*) \leftarrow \{p' \in N_{\text{out}}(p^*) : F_{p'} \cap F_q \neq \emptyset, p' \notin \mathcal{V}\}$
5 $\mathcal{L} \leftarrow \mathcal{L} \cup N'_{\text{out}}(p^*)$
if $|\mathcal{L}| > L$ **then**
6 Update $\mathcal{L}$ with the closest L nodes to x_q.
return [k NNs from $\mathcal{L}$; $\mathcal{V}$]

Algorithm 4: FilteredVamana Indexing Algorithm

Data: Database P with n points where i-th point has coords x_i, parameters α, L, R.
Result: Directed graph G over P with out-degree $\leq R$.
begin
1 Initialize G to an empty graph
2 Let s denote the medoid of P
3 Let $\text{st}(f)$ denote the start node for filter label f for every $f \in F$
4 Let σ be a random permutation of $[n]$
5 Let F_x be the label-set for every $x \in P$
foreach $i \in [n]$ **do**
6 Let $S_{F_{x_{\sigma(i)}}} = \{\text{st}(f) : f \in F_{x_{\sigma(i)}}\}$
7 Let $[\emptyset; \mathcal{V}_{F_{x_{\sigma(i)}}}] \leftarrow$ `FilteredGreedySearch`$(S_{F_{x_{\sigma(i)}}}, x_{\sigma(i)}, 0, L, F_{x_{\sigma(i)}})$
8 $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}_{F_{x_{\sigma(i)}}}$
9 Run `FilteredRobustPrune`$(\sigma(i), \mathcal{V}_{F_{x_{\sigma(i)}}}, \alpha, R)$ to update out-neighbors of $\sigma(i)$.
foreach $j \in N_{\text{out}}(\sigma(i))$ **do**
10 Update $N_{\text{out}}(j) \leftarrow N_{\text{out}}(j) \cup \{\sigma(i)\}$
if $|N_{\text{out}}(j)| > R$ **then**
11 Run `FilteredRobustPrune`$(j, N_{\text{out}}(j), \alpha, R)$ to update out-neighbors of j.

Filtered-DiskANN（4/4）

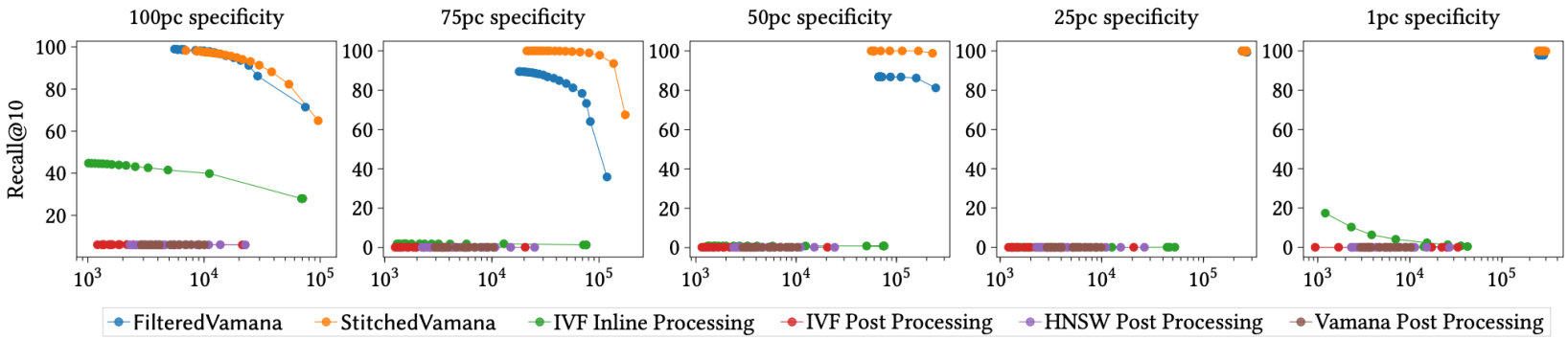


Figure 1: Turing dataset: QPS (x-axis) vs recall@10 for various algorithms with filters of 100, 75, 50, 25 and 1 percentile specificity.

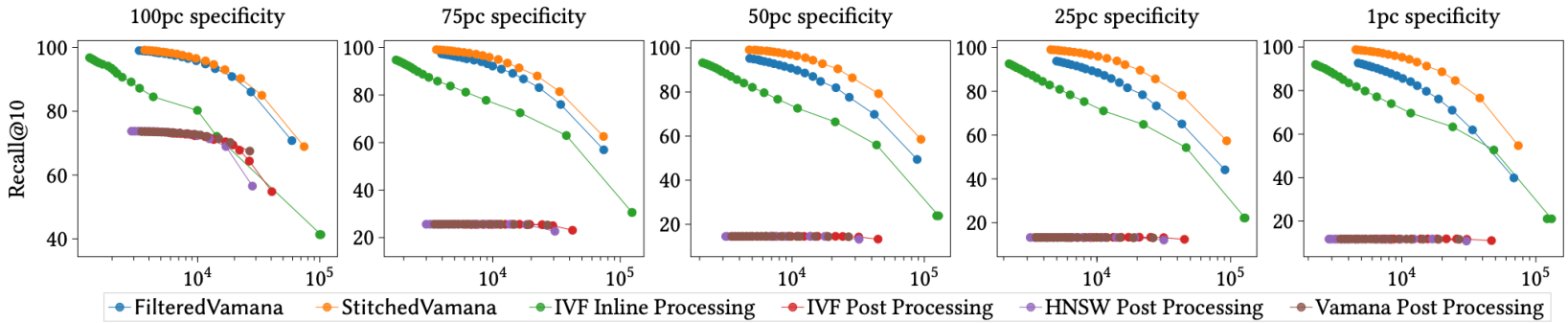


Figure 2: Prep dataset: QPS (x-axis) vs recall@10 for various algorithms with filters of 100, 75, 50, 25 and 1 percentile specificity.

Summary

1. ANNS 定义及其研究背景.
2. ANNS 索引算法（Graph/Tree/Hash/PQ）.
3. ANNS 混合检索: AnalyticDB-V, Filtered-DiskANN.
4. 流式 ANNS: AnalyticDB-V, FreshDiskANN.

End

Thank you!

Q&A

Reference

- https://zhuanlan.zhihu.com/p/686251186 向量相似性搜索技术-扫盲篇（LLM, RAG 背后的支持技术）
- https://www.slideshare.net/slideshow/approximate-nearest-neighbor-methods-and-vector-models-nyc-ml-meetup/53177248
- W. Wei, et al., VLDB 2020. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data.
- A. Gollapudi, et al., WWW 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters.
- A. Singh, S. J. Subramanya, R. Krishnaswamy, et al., arXiv 2021. FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search.