香港中文大學
The Chinese University of Hong Kong

西安電子科技大學
XIDIAN UNIVERSITY

香港科技大學(廣州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

VLDB 2026 TUTORIAL

Advances of Query Processing in Vector Databases

Jiadong Xie[1] Yingfan Liu[2] Jeffrey Xu Yu[3]

[1] The Chinese University of Hong Kong
[2] Xidian University
[3] The Hong Kong University of Science and Technology (Guangzhou)

Tutorial Overview

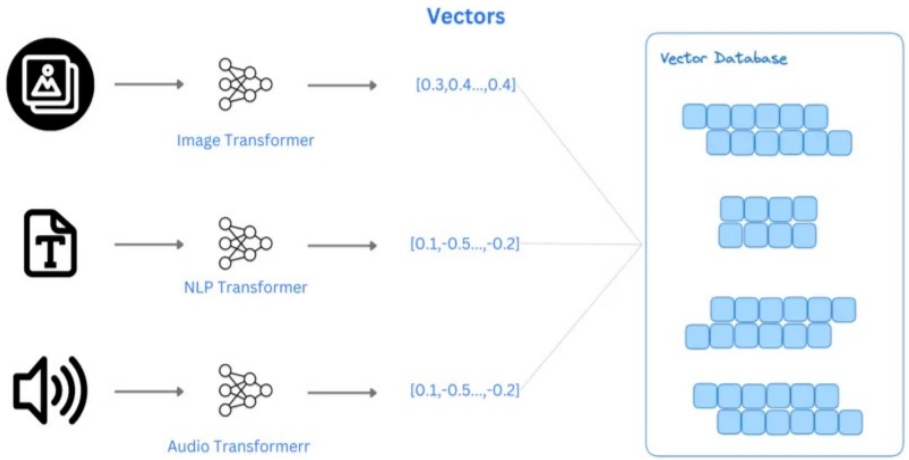
- **Part 1** [10min, 13:45--13:55]: Background and Motivation
- **Part 2** [80min, 13:55--15:15]: Similarity Search
- **Part 3** [20min, 15:45--16:05]: Multi-Similarity Search
- **Part 4** [45min, 16:05--16:50]: Filtered Similarity Search
- **Part 5** [15min, 16:50-17:05]: Similarity Join
- **Part 6** [10min, 17:05-17:15]: Open Challenges

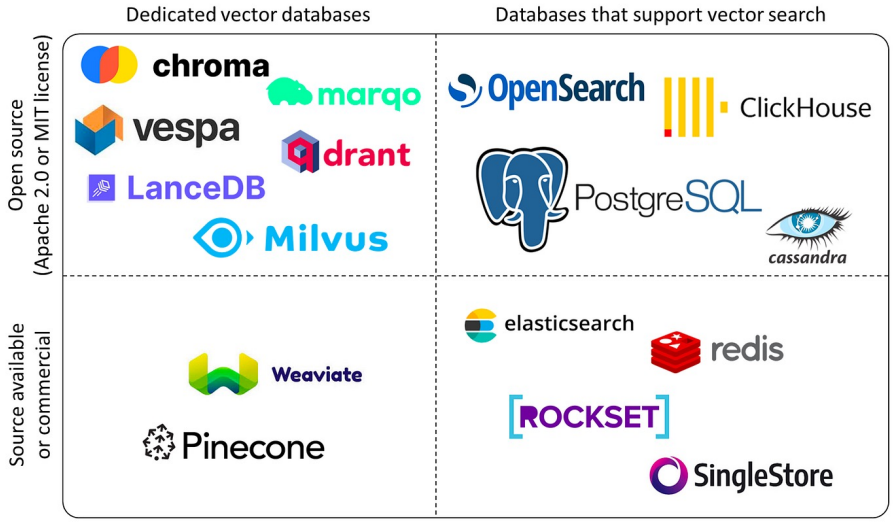
Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

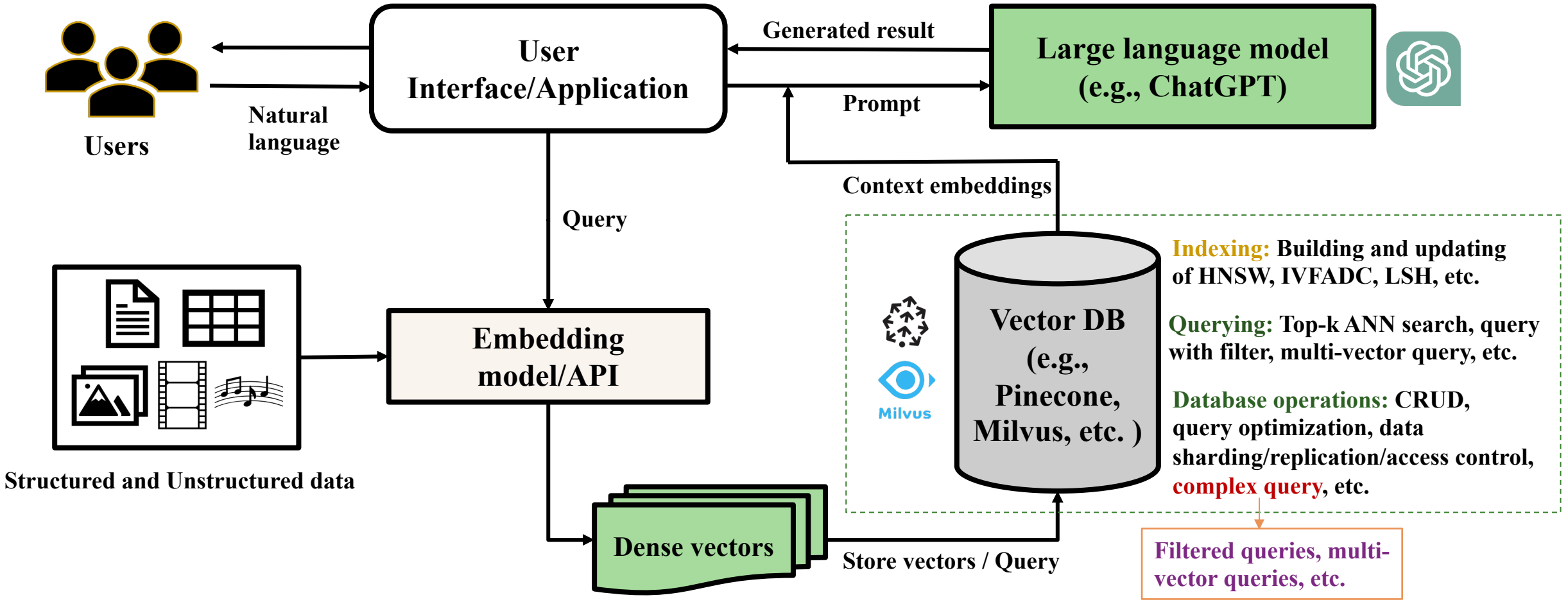
Vector Databases

- **The Vector Data**
  - The success of deep learning turns **everything** into vectors.
  - Vector is a **universal data representation** connecting different modalities and domains, and could be a foundation for cross-domain innovations.
- **The Vector Databases**
  - Store vectors with identifiers, metadata, and often the original objects.
  - Similarity search over objects/texts is implemented as **similarity search** over their **vectors** representations.
  - More than a container of vectors, it must support indexing, updates, distributed execution, and complex query semantics.



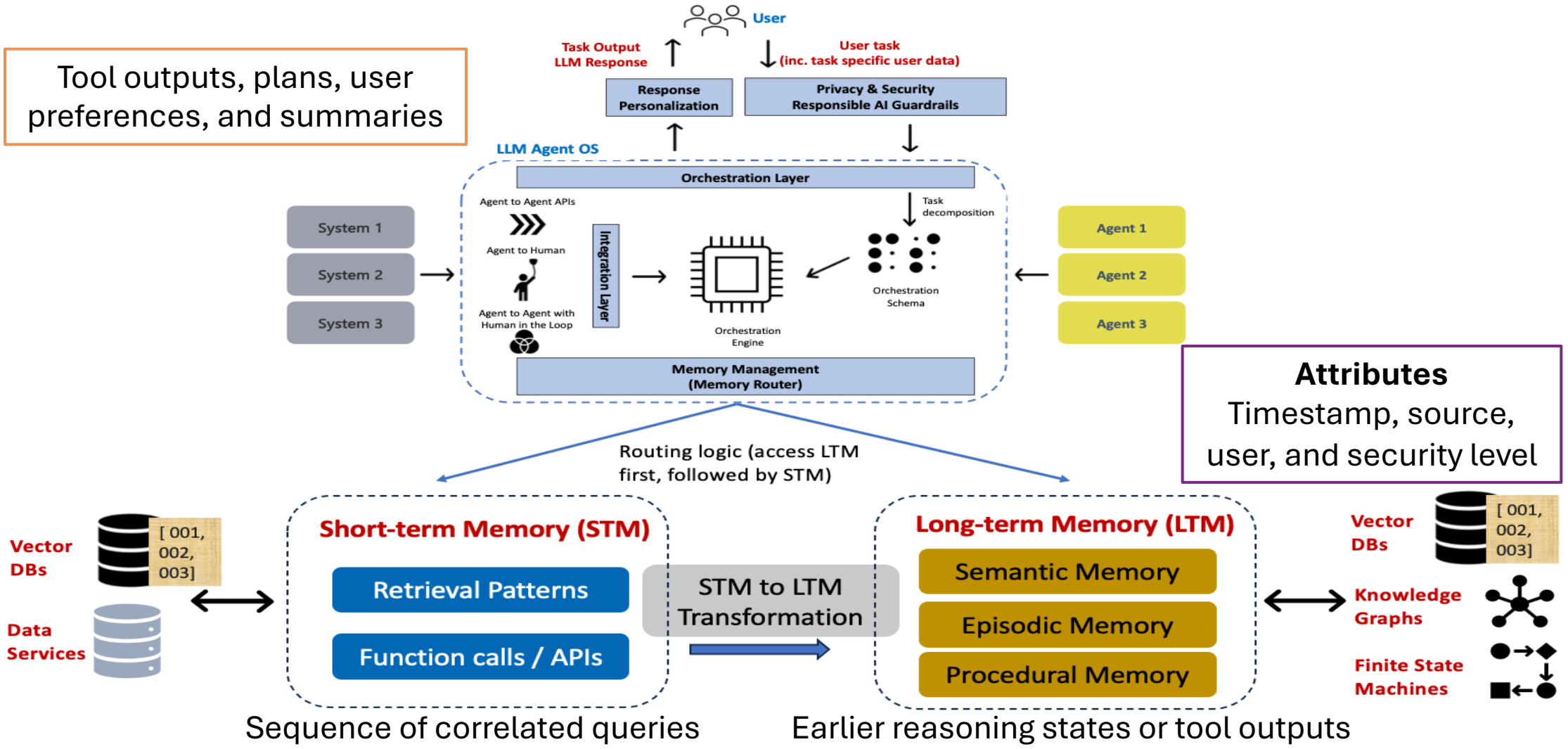


Vector Databases in the Era of LLMs



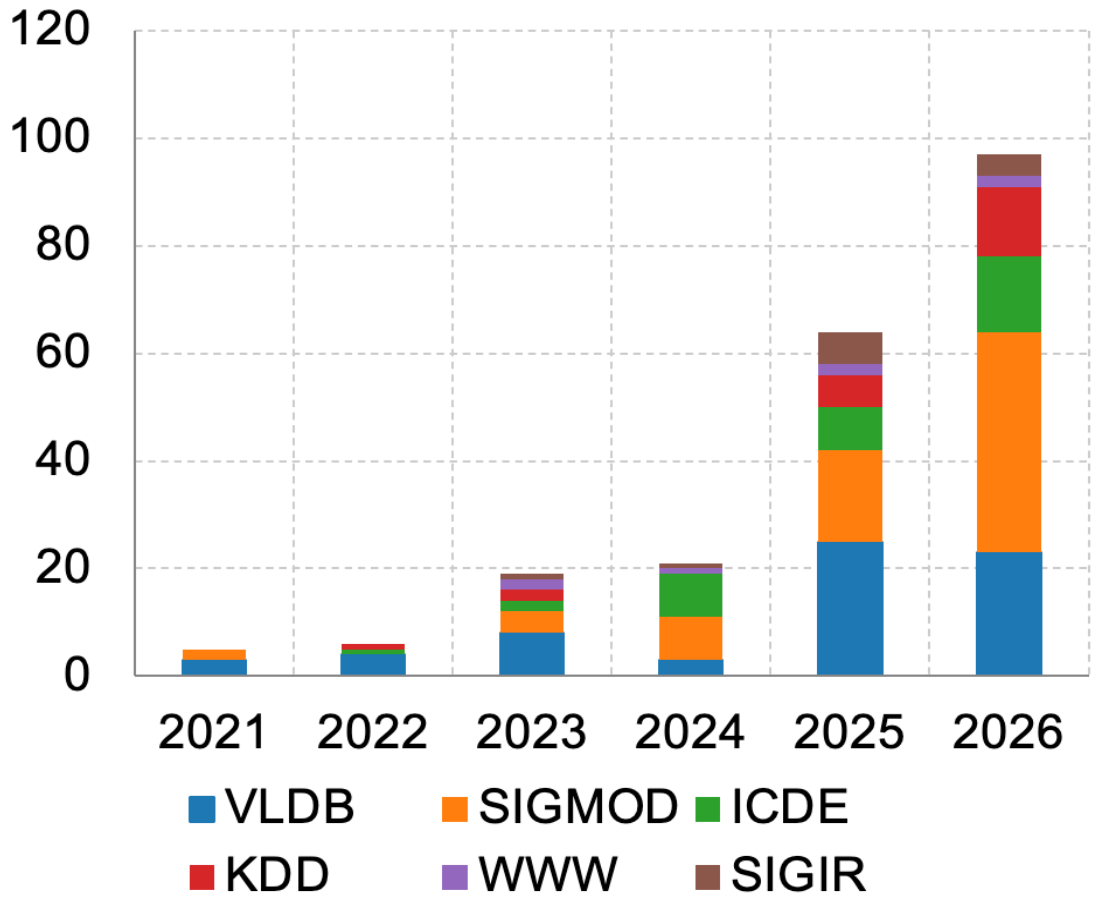
Taken from a talk given by Prof. Xiaofang Zhou.

Vector Databases in Agent Memory Management



Taken from https://ai.gopubby.com/long-term-memory-for-agentic-ai-systems-4ae9b37c6c0f.

Publications in Recent Communities

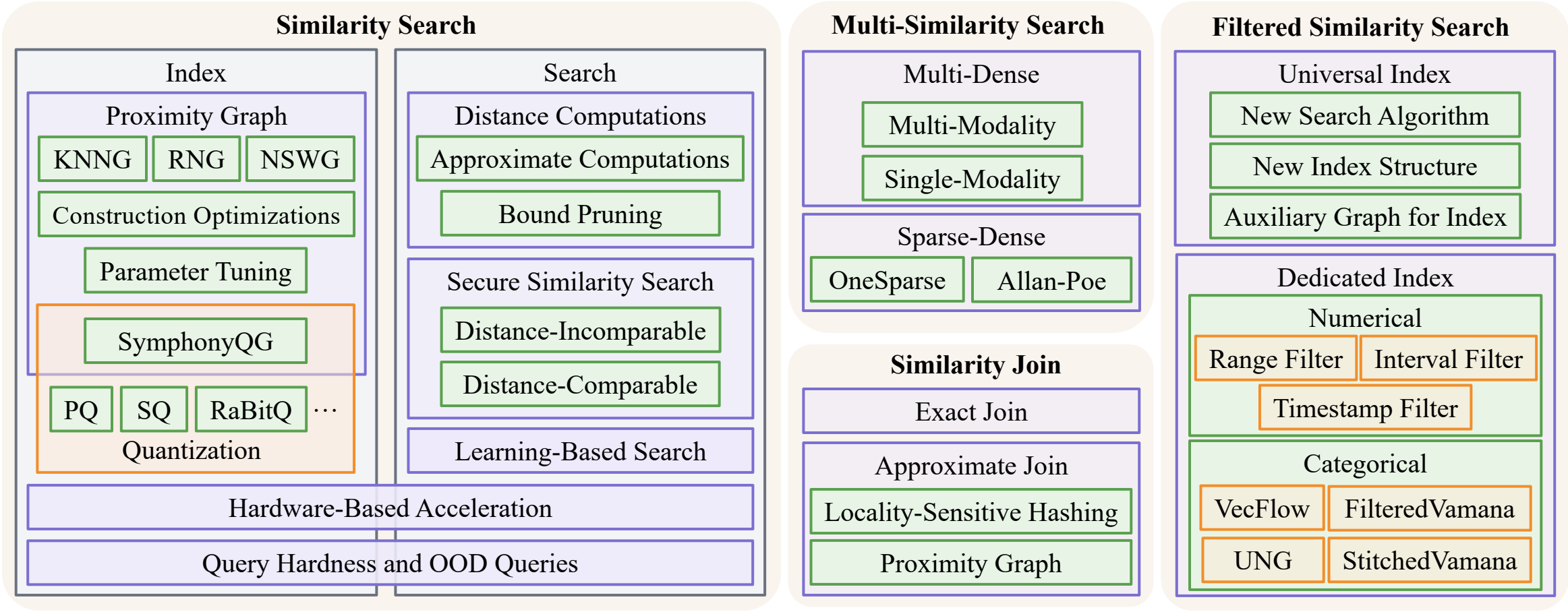


Recent Diversifying Topics

This growth is precisely why a structured survey is now useful.

Statistics with help from CodeX. Until 18 Aug 2026.

Tutorial Conceptual Map



No longer **separate islands**: many new systems try to share indexes, search logic, and cost models across them.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
  - **Part 2.1**: Proximity Graph Based Approaches
  - **Part 2.2**: Quantization-Based Approaches
  - **Part 2.3**: Distance Computation Acceleration
  - **Part 2.4**: Hard and Out-of-Distribution Queries
  - **Part 2.5**: Secure Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Similarity Search

- **Nearest Neighbor (NN) Search**
  - Similarity search on such objects/texts is to similarity search on vectors in the vector database.
  - **Example**: find the top 4 similar coats to the ${input_image}

${input_image}

```
SELECT id
FROM clothes_store
ORDER BY DISTANCE(image, ${input_image})
LIMIT 4
```

Similarity Search

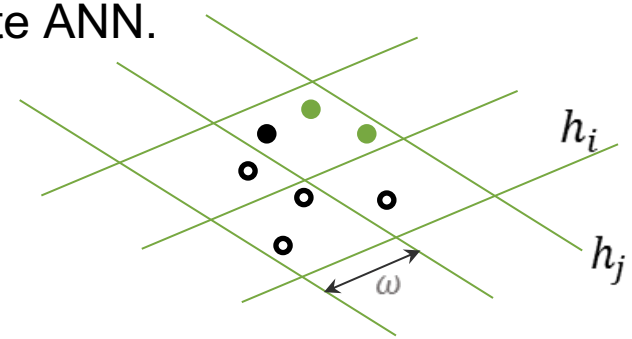
- **Nearest Neighbor (NN) Search**
  - Given a dataset $D \subseteq R^d$ with n points (vectors), and a query $q \in R^d$, a nearest neighbor search is to find a point $p \in D$, which is closest to q.
  - Finding NN often expensive due to the curse of dimensionality; in many practical settings, the cost approaches a near-linear scan.

- **Approximate Nearest Neighbor Search (ANNS)**
  - Given a data set $D \subseteq R^d$ with n points (vecters) and a query $q \in R^d$, ANNS is to find a point $p \in D$, which is sufficiently close to q.
  - ANNS trades off accuracy for efficiency, and is widely used in the real world.

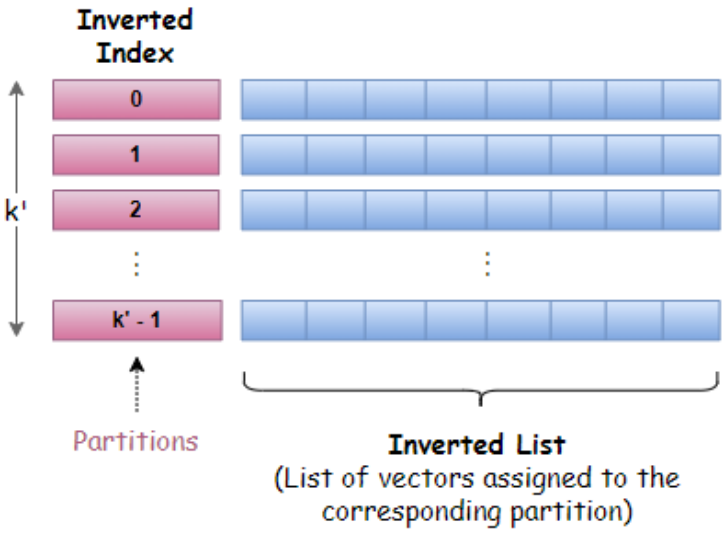
- The similarity search is to find top-1 or top-k nearest neighbors, in general.

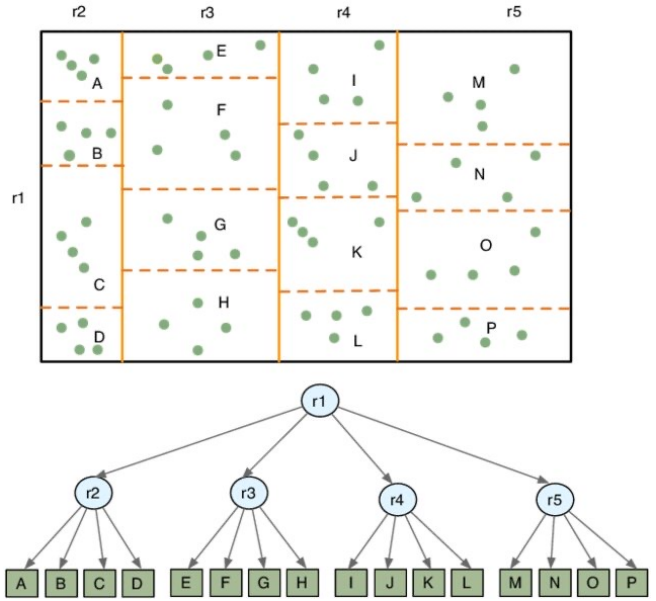
Index Families for ANN Search

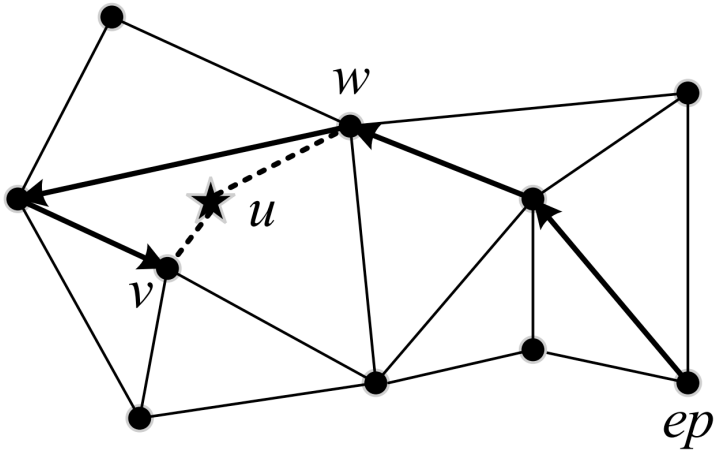
- To employ an index to retrieve the promising vectors and find accurate ANN.
  - Inverted-index based index structures
  - Tree based index structures
  - Locality sensitive hashing based index structures
  - **Proximity graph based index structures**
  - **Quantization based index structures**



G_1





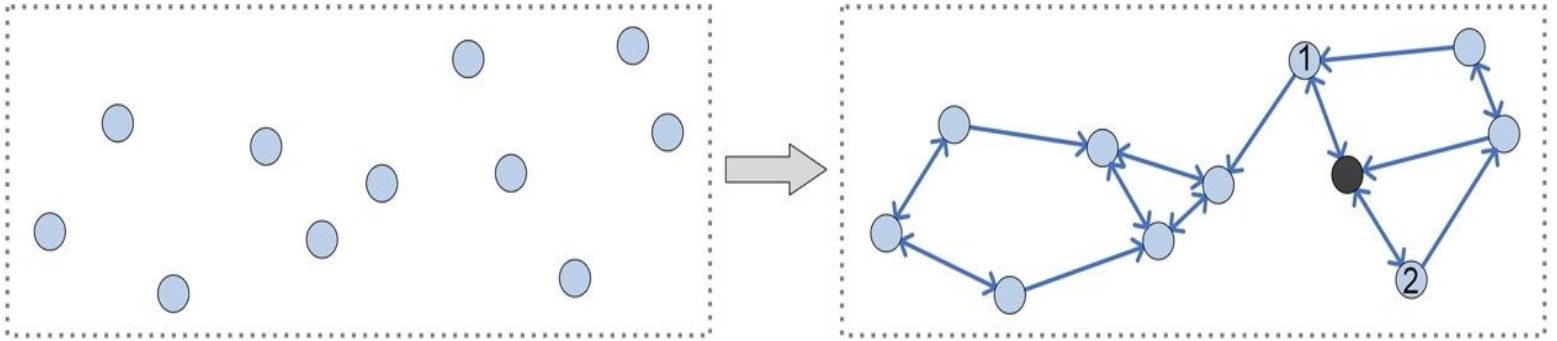


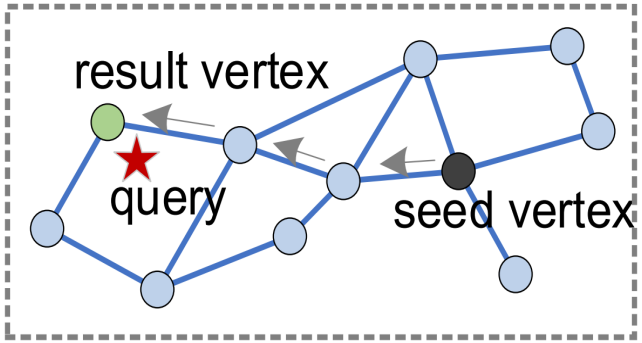
Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
  - **Part 2.1**: Proximity Graph Based Approaches
  - **Part 2.2**: Quantization-Based Approaches
  - **Part 2.3**: Distance Computation Acceleration
  - **Part 2.4**: Hard and Out-of-Distribution Queries
  - **Part 2.5**: Secure Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Proximity Graph-Based Indexes

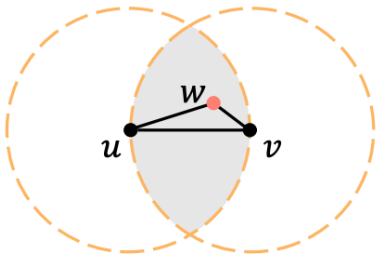
- A proximity graph (PG) $G = (V, E)$ of D is a directed graph, where each node in V uniquely represents a vector (i.e., data point) in D, and two nodes are connected by an edge in E if their corresponding vectors are close to each other.



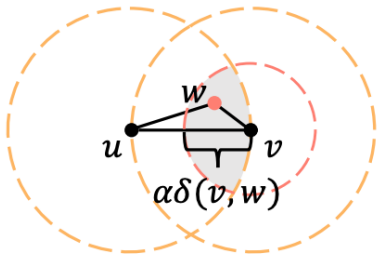


Edge Selection Strategies

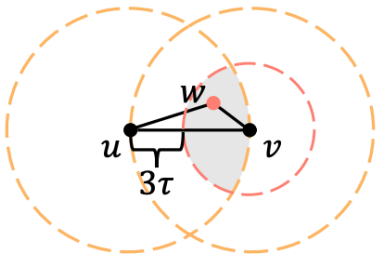
- Different approaches have the same vertex set but different edge sets, depending on their respective **edge selection strategies**.
  - ***k*-ANN pruning**: The simplest pruning strategy is to retain, for each node u, only the nearest k candidates according to their distances, i.e., a top-k nearest-neighbor selection [1]
  - Other pruning strategies are proposed based on the k-ANN pruning



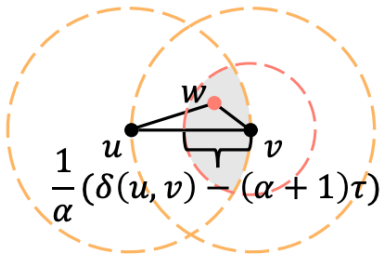
RNG Pruning [2][3]



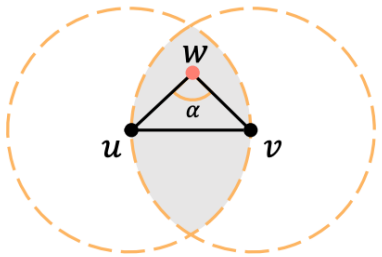
Robust Pruning [4]



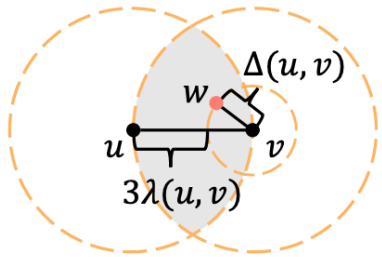
τ-Pruning [5]



α-CG Pruning [6]



α-Pruning [7]

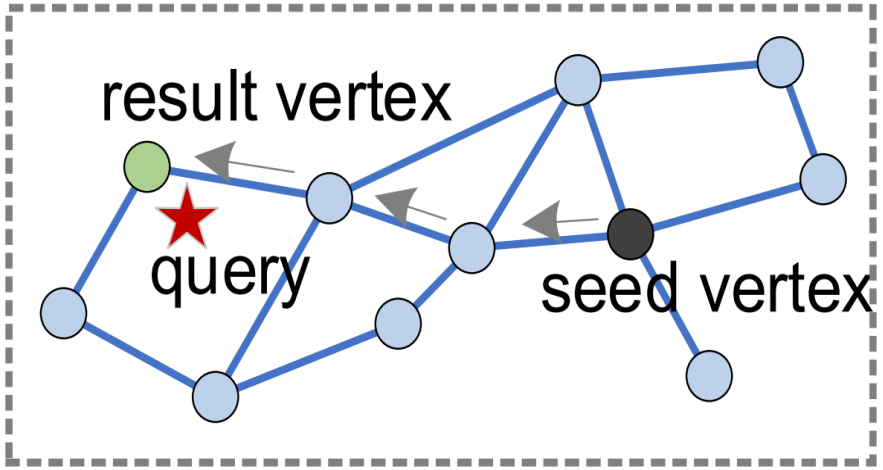


ALMG Pruning [8]

[1] Dong et al., Efficient k-nearest neighbor graph construction for generic similarity measures. **WWW 2021**.
[2] Malkov et al., Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. **TPAMI 2020**.
[3] Fu et. al., Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph . **VLDB 2019**.
[4] Subramanya et al., DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. **Neuips 2019**.
[5] Peng et al., Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. **SIGMOD 2023**.
[6] Li et al., Fast-Convergent Proximity Graphs for Approximate Nearest Neighbor Search. **SIGMOD 2026**.
[7] Yang et al., Revisiting the Index Construction of Proximity Graph-Based Approximate Nearest Neighbor Search. **VLDB 2025**.
[8] Xie et al., Graph Based K-Nearest Neighbor Search Revisited. **TODS 2025**.

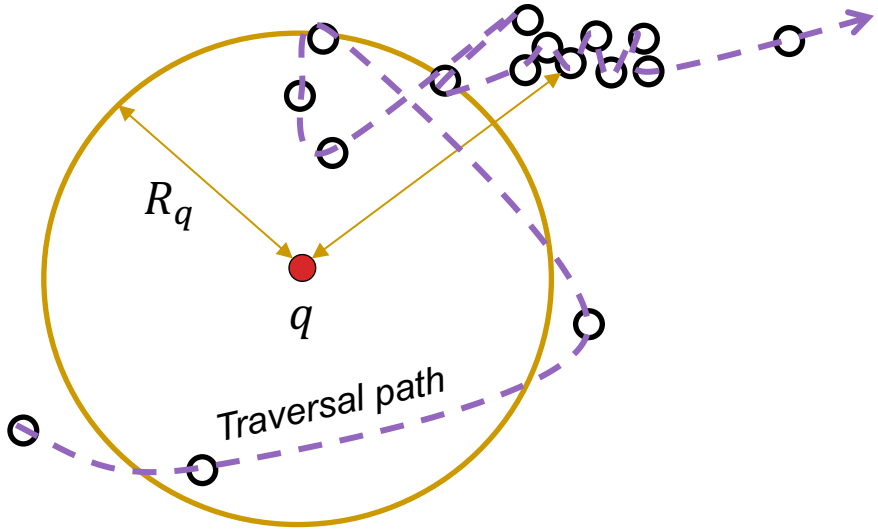
Beam Search

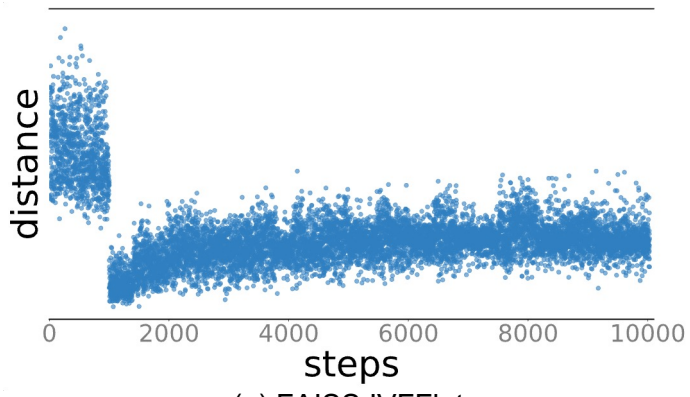
- The PG construction algorithms vary, but many different PGs share the same search algorithm.
- The search starts at the entry point (seed vertex)
- The algorithm iteratively finds the closest but unexpanded neighbor u from $pool$ and expands u to refine $pool$.
  - each neighbor $v \in N_G(u)$ is treated as a k-ANN candidate of q
  - further verified by an expensive distance computation to refine $pool$
- A sorted array $pool$ is used to maintain the currently L-closest neighbors found.
  - L is the beam width, limiting how many candidates are retained or actively explored.
  - $L = 1$, beam search becomes the greedy routing.



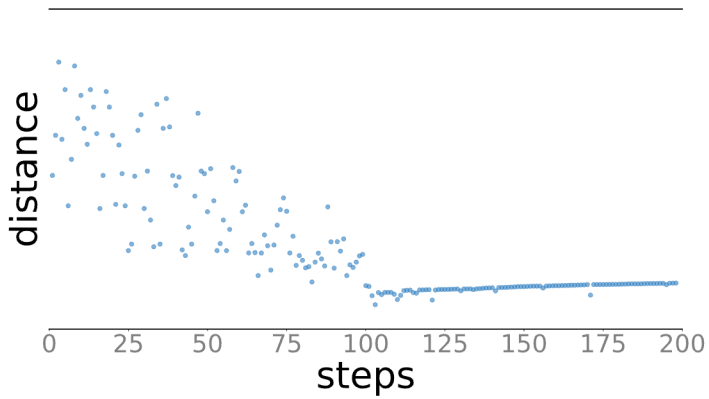
Two Phases in Beam Search

- **Vector index traversal shows Relaxed Monotonicity w/ a *two-phase pattern***
  - Phase 1: Approaches target neighborhood with large distance oscillations
  - Phase 2: Departs from target neighborhood approximately and steadily





(a) FAISS IVFFlat



(b) HNSW

[1] VBASE: Zhang et al., VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. **OSDI 2023**.

Search on PGs: Theoretical Guarantee

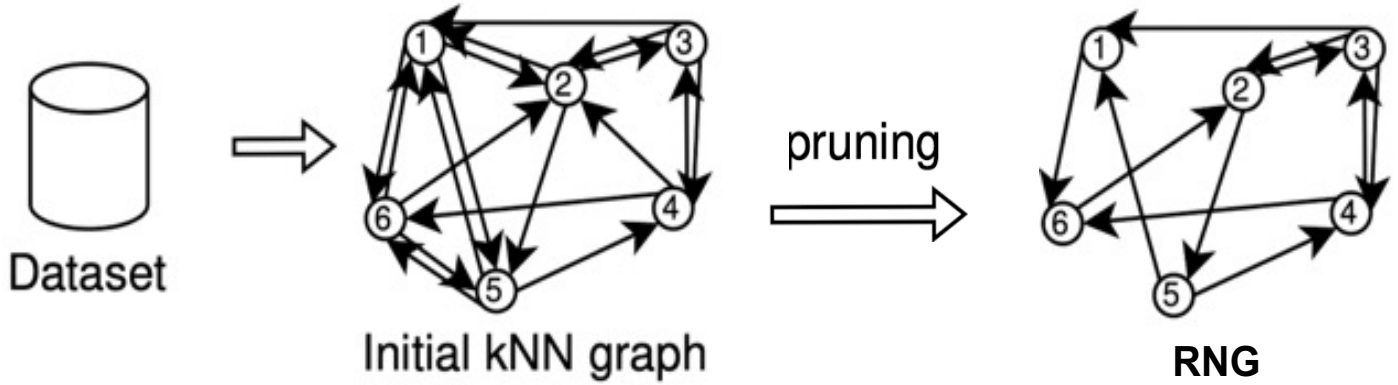
- DG can find 1-NN of a query point q in $O(n)$ time, which is inefficient.
- To achieve high efficiency, the majority of PG-based approaches, e.g., HNSW, NSG, SSG, DPG, do not have a guarantee in theory to find 1-NN.
- MRNG, Vamana guarantees to find 1-NN of a query point q on the condition that the query point q is a point in D $(q \in D)$.
- FANNG, τ-MG, α-CG can find 1-NN p_1 of q if $\delta(p_1,q) < \tau$ in theory, but τ needs to predetermine the value of τ.
- Vamana, α-CG further ensure the identification of the $\left(\frac{\alpha+1}{\alpha-1}+\epsilon\right)$-ANN when $\delta(p_1,q) > 0$ and $\delta(p_1,q) > \tau$, respectively.
- LMG ensure the finding of both the 1-NN and k-NN without any constraints on $\delta(p_1,q)$.

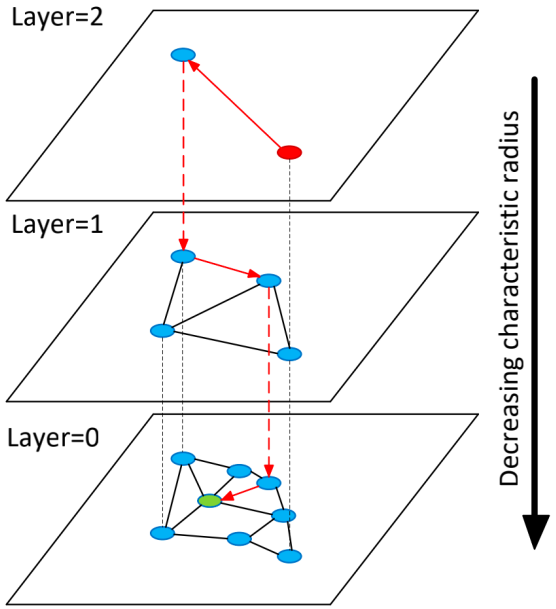
- MRNG, τ-MG, α-CG, Vamana, and LMG all need near $O(n^2)$ for index construction time or index size.

[1] DG: Govinda D. Kurup. Database Organized on the Basis of Similarities with Applications in Computer Vision. **1992**.
[2] HSNW: Malkov et al., Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. **TPAMI 2020**.
[3] SSG: Fu te al., High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. **TPAMI 2022**.
[4] DPG: Li et. al., Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. **TKDE 2019**.
[5] NSG/MRNG: Fu et. al., Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph . **VLDB 2019**.
[6] Vamana: Subramanya et al., DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. **Neuips 2019**.
[7] FANNG: FANNG: Fast Approximate Nearest Neighbour Graphs. **CVPR 2016**.
[8] τ-MG: Peng et al., Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. **SIGMOD 2023**.
[9] α-CG: Li et al., Fast-Convergent Proximity Graphs for Approximate Nearest Neighbor Search. **SIGMOD 2026**.
[10] LMG: Xie et al., Graph Based K-Nearest Neighbor Search Revisited. **TODS 2025**.

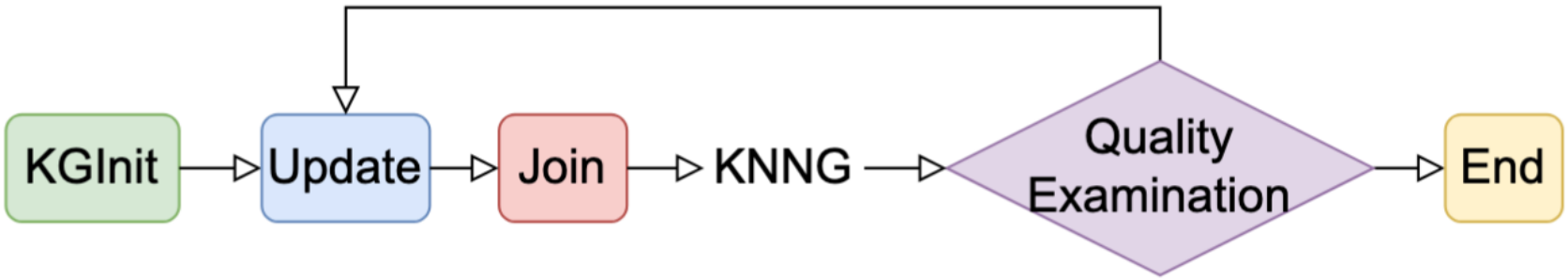
Three Categories for Index Construction

- k-Nearest Neighbor Graph (KNNG)
- Relative Neighborhood Graph (RNG)
  - Based on KNNG, it removes the longest edge in every possible triangle
- Navigable Small World Graph (NSWG)
  - Construction involves incrementally node-by-node insertion, where each node is connected to its k-ANN in the current incomplete graph





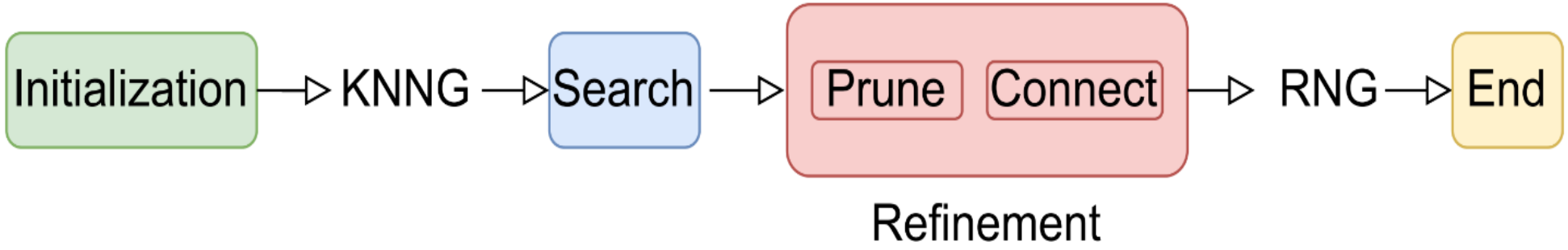
KGraph: Index Construction of KNNG



- **KGraph Initialization** Phase: a random graph
- **Update** Phase
  - For each $u \in D$, update its out-neighbors using its in-neighbors
- **Join** Phase
  - Apply the principle that neighbors of a node's neighbors are likely to be its neighbors
  - For each $u \in D$, update its out-neighbors using its 2-hop neighbors

[1] Dong et al., Efficient k-nearest neighbor graph construction for generic similarity measures. **WWW 2021**.

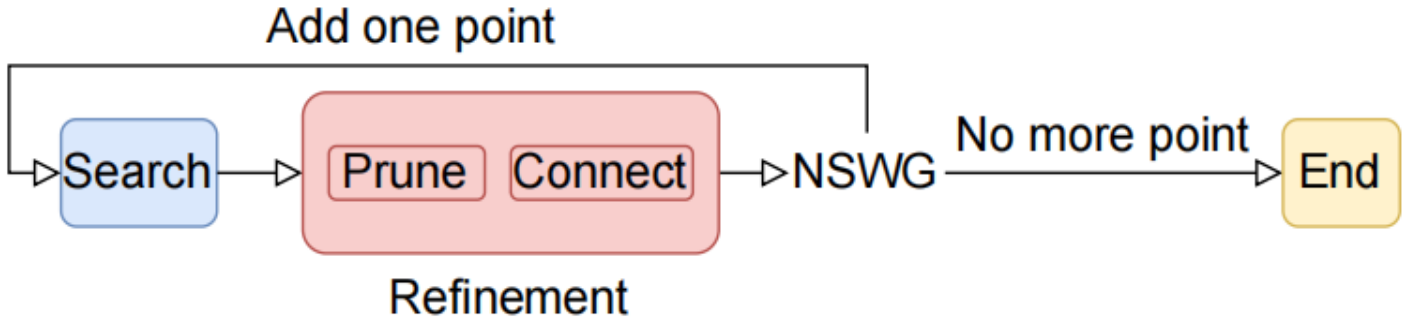
NSG: Index Construction of RNG

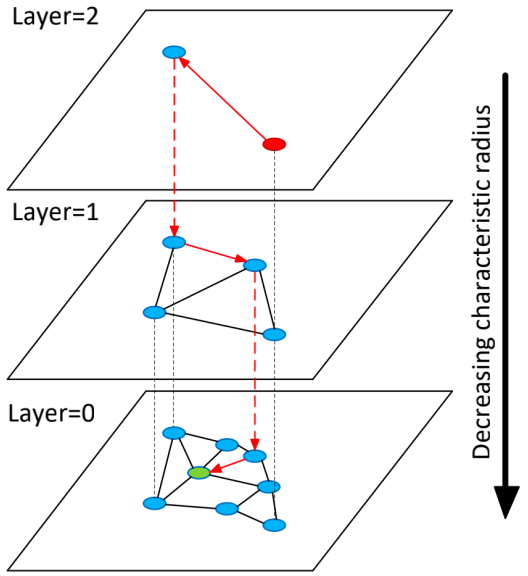


- **Initialization** Phase
- **Search** Phase
  - For each $u \in D$, perform k-ANN search on KNNG in order to obtain the candidate set $C(u)$.
- **Refinement** Phase
  - Remove redundant neighbors in the set $C(u)$ via a pruning strategy to obtain $N_G(u)$ with the constraint $|N_G(u) \leq M|$, where M is a specific threshold.
  - To improve the graph connectivity, unidirectional edges are added between u and each $v \in N_G(u)$.

[1] Fu et. al., Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph . **VLDB 2019**.

HNSW: Index Construction of NSWG





- **Initializing** the graph with a single point, for each remaining point
  - To determine its highest layer l using an exponentially decaying probability distribution
- **Search** Phase
  - To perform k-ANN search on current index in order to obtain the candidate set $C(u)$.
- **Refinement** Phase
  - To Remove redundant neighbors in the set $C(u)$ via a pruning strategy to obtain $N_G(u)$ with the constraint $|N_G(u) \leq M|$, where M is a specific threshold.
  - To improve the graph connectivity, unidirectional edges are added between u and each $v \in N_G(u)$.

[1] Malkov et al., Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. **TPAMI 2020**.

Partition-Based Index Construction

- This line first partitions the dataset and then builds local graphs on the partitions
  - Reduce construction cost and improve search efficiency by exploiting **intra-partition** locality
  - Preserving global navigability through **inter-partition** connections.
- Tree based partitioning: NGT [1]
- Hierarchical clustering based partitioning: HCNNG [2]
- partitions the dataset into multiple overlapping subsets: CSPG [3]

[1] Iwasaki et al., Optimization of Indexing Based on k-Nearest Neighbor Graph for Proximity Search in Highdimensional Data. **Arxiv 2018**.
[2] Muñoz et al., Hierarchical Clustering-Based Graphs for Large Scale Approximate Nearest Neighbor Search. Pattern **Recognit. 2019**.
[3] Yang et al., CSPG: Crossing Sparse Proximity Graphs for Approximate Nearest Neighbor Search. **NeurIPS 2024**.

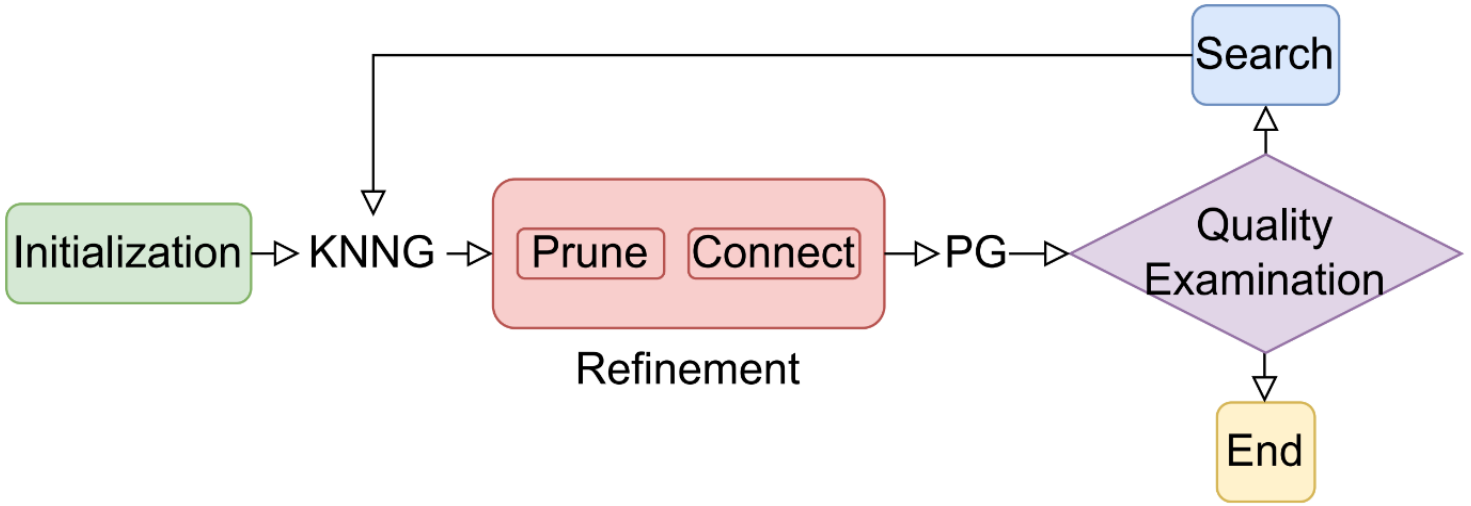
Parallel and Hardware-Aware Index Construction

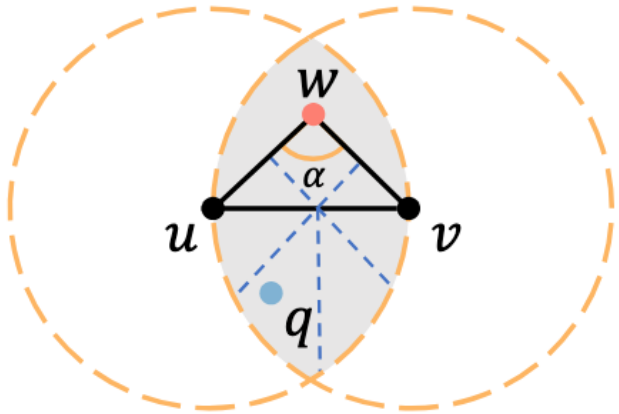
- One direction for index build acceleration is to exploit the massive parallelism of CPUs/GPUs.

Construction Strategies	Description	Representative Instances
Divide-and-conquer	Recursively generate subgraphs and merge	ELPIS (CPU) [1] GGNN (GPU) [2]
Increment-based	Continuously insert data into the index	SONG (GPU) [3] GANNS (GPU) [4] LSH-APG (CPU) [5] ParlayANN (CPU) [6]
Refinement-based	Initialize a graph and prune it to refine	CAGRA (GPU) [7] Tagore (GPU) [8] RNN-Descent (CPU) [9]

[1] Azizi et al. Elpis: Graph-based similarity search for scalable data science. **VLDB 2023**.
[2] Groh et al., GGNN: Graph-based gpu nearest neighbor search. **IEEE Big Data 2022**.
[3] Zhao et al., SONG: Approximate nearest neighbor search on gpu. **ICDE 2020**.
[4] Yu et al., GPU-accelerated proximity graph approximate nearest Neighbor search and construction. **ICDE 2022**.
[5] Zhao et al., Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. **VLDB 2023**.
[6] Manohar et al., ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search Algorithms. **PPoPP 2024**.
[7] Ootomo et al., CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs. **ICDE 2024**.
[8] Li et al., Scalable Graph Indexing using GPUs for Approximate Nearest Neighbor Search. **SIGMOD 2026**.
[9] Ono et al., Relative NN-Descent: A Fast Index Construction for Graph-Based Approximate Nearest Neighbor Search. **MM 2023**.

FastNSG/FastHNSW



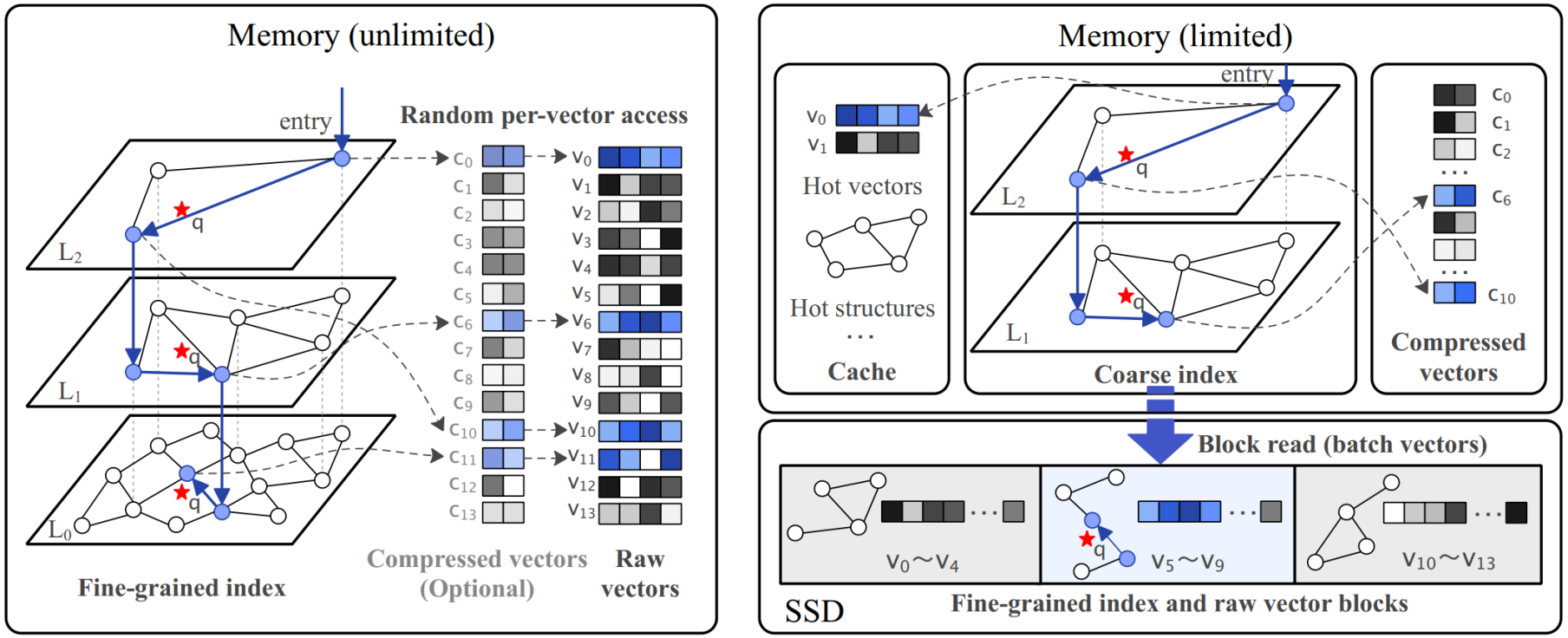


- Replace the **search-before-refinement** scheme with a new **refinement-before-search** scheme.
- To ensure high-quality k-ANN result
  - **α-Pruning Strategy:** When α becomes larger, the higher probability that search-before-refinement and refinement-before-search have the same k-ANN result
    - An edge (u, v) exists in the graph only if there is no edge (u, w) in the graph where
      - $dist(u, w) < dist(u, v)$,
      - $dist(v, w) < dist(u, v)$,
      - and $\angle uwv > \alpha$.

[1] Yang et al., Revisiting the Index Construction of Proximity Graph-Based Approximate Nearest Neighbor Search. **VLDB 2025.**

Index Size Pressure on PGs

- **Proximity graph** significantly **increases memory requirements and pressure**.
  - One direction improves scalability by leveraging external storage or richer memory hierarchies



(a) Memory-Resident VS (b) Memory-SSD VS

[1] Song et al. Vector Search for the Future: From Memory-Resident, Static Heterogeneous Storage, to Cloud-Native Architectures. **SIGMOD 2026**.

Proximity Graph Index Maintenance

- **Graph** based index structures
  - Difficult to **maintain the index**, especially for deletion data points

  - NSWG, such as HNSW and NSW, originally support data points incremental insertion
  - Out-of-place update with periodical global rebuilds, e.g., ADBV, Milvus.
    - Accumulate and index new vectors in a secondary in-memory index
    - Periodically merge into the base index by global rebuilds
    - To reduce global rebuild costs, SimJoin leverages approximate similarity join to accelerate rebuilding

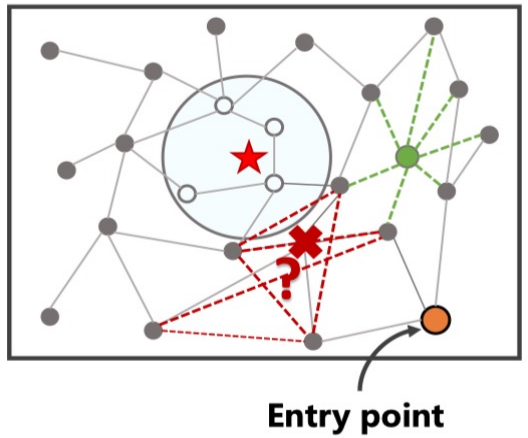
	Memory	CPU	Time
DiskANN	1100 GB	32 cores	2 days
	64GB	16 cores	5 days
SPANN	260 GB	45 cores	4 days

Global rebuild costs

[1] ADBV: Wei et al., AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data. **VLDB 2020**.
[2] Milvus: Wang et al., Milvus: A Purpose-Built Vector Data Management System. **SIGMOD 2021**.
[3] SimJoin: Xie et al., Fast Approximate Similarity Join in Vector Databases. **SIGMOD 2025**.

Proximity Graph Index Maintenance

- Many approaches attempt to do **in-place update** in graph-based indices
  - FreshDiskANN handles deletions by locally reconnecting the neighbors of the deleted point
    - it treats them as candidate neighbors for one another and reapplies pruning to restore connectivity.
  - IP-DiskANN repairs each affected out-neighbor using only a restricted nearby candidate set rather than exhaustive reconnection thereby substantially reducing repair cost.
  - Wolverine++ improves deletion repair by expanding the candidate space from 1-hop to 2-hop neighbors.
  - PRO-HNSW reconnects disconnected components via limited-hop BFS.



[1] FreshDiskANN: Singh et al., FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search. **ArXiv 2021**.
[2] IP-DiskANN: Xu et al., In-Place Updates of a Graph Index for Streaming Approximate Nearest Neighbor Search. **Arxiv 2025**.
[3] Wolverine++: Li et al., Wolverine: Highly Efficient Monotonic Search Path Repair for Graph-based ANN Index Updates. **VLDB 2025**.
[4] PRO-HNSW: Jin et al., PRO-HNSW: Proactive Repair and Optimization for High-Performance Dynamic HNSW Indexes. **ICDE 2026**.

Parameter Tuning for Index Construction

- The performance of PG-based approaches is highly sensitive to **hyperparameters**, such as the degree limit, construction beam width, and search beam width.
  - These parameters jointly determine the trade-off between index cost and search performance.
- Grid search evaluates configurations on a predefined grid
- Random search samples directly from the search space
- VDTuner formulates tuning as a constrained optimization problem under a target recall
  - It uses constrained Bayesian optimization to iteratively recommend promising configurations
- PGTuner targets automatic and transferable tuning for PGs
  - Combines a pre-trained query-performance predictor with a deep reinforcement learning-based recommendation model
  - Supports efficient retuning through out-of-distribution detection and deep active learning
- FastPGT observes that the main bottleneck lies in repeatedly building and evaluating PGs
  - It therefore builds multiple PGs simultaneously, it shares repeated computations across configurations while building/evaluating multiple graphs

[1] Grid search: Liashchynskyi et al., Grid search, random search, genetic algorithm: a big comparison for NAS. **Arxiv 2019**.
[2] Random search: Bergstra et al., Random search for hyper-parameter optimization. **Journal of Machine Learning Research 2012**.
[3] VDTuner: Yang et al., Vdtuner: Automated performance tuning for vector data management systems. **ICDE 2024**.
[4] PGTuner: Duan et al., PGTuner: An Efficient Framework for Automatic and Transferable Configuration Tuning of Proximity Graphs. **SIGMOD 2026**.
[5] FastPGT: Zhou et al., Fast Tuning the Index Construction Parameters of Proximity Graphs in Vector Databases. **Arxiv 2026**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
  - **Part 2.1**: Proximity Graph Based Approaches
  - **Part 2.2**: Quantization-Based Approaches
  - **Part 2.3**: Distance Computation Acceleration
  - **Part 2.4**: Hard and Out-of-Distribution Queries
  - **Part 2.5**: Secure Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

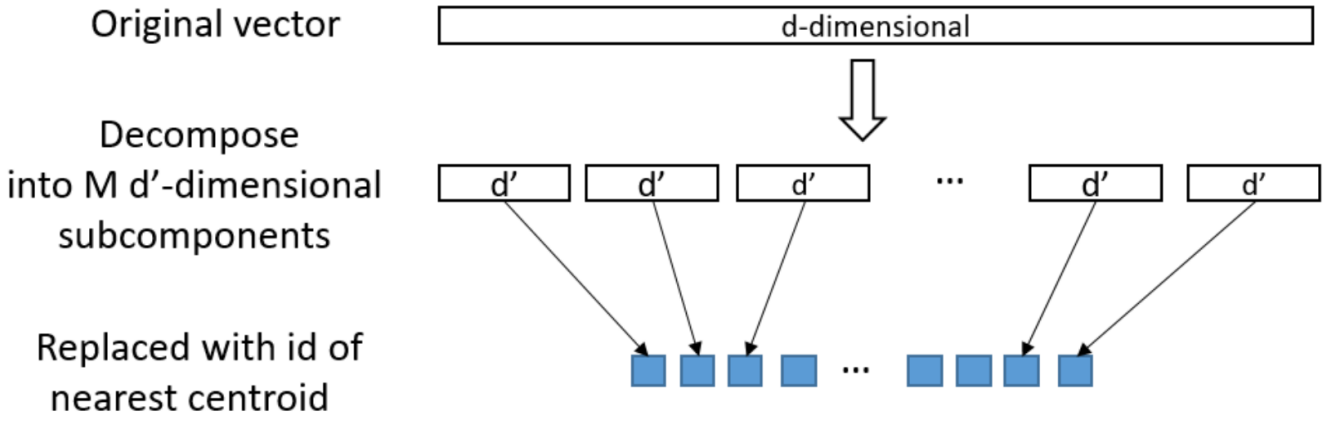
Quantization-based ANN Search Approaches

- **Graph**-based index structures
  - They require full-precision vectors in memory and thus hard to process billion-scale datasets.
- **Quantization**-based Approaches
  - They compress each high-dimensional vector into short codes (e.g. 128/256 bits)
  - They significantly **reduce the memory usage**, and enable **approximate distance computations**
    - PQ is the **classical** method and provides a biased distance estimator without theoretical error bound
    - RaBitQ guarantees a sharp **theoretical error bound** and provides **good empirical accuracy** simultaneously.
    - RaBitQ+ extends RabitQ in support of achieving **higher accuracy by using more space**, which achieves the asymptotic optimality in terms of the trade-off between space and error bounds.
    - SuCo provides theoretical guarantees on the quality of its results, and can achieve efficient and accurate ANN search.

[1] PQ: Jegou et al., Product quantization for nearest neighbor search. **IEEE TPAMI 2011**.
[2] RaBitQ: Gao et al., RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. **SIGMOD 2024**.
[3] RaBitQ (ext): Gao et al., Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. **SIGMOD 2025**.
[4] SuCo: Wei et al., Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. **SIGMOD 2025**.

Product Quantization

- Vector Encoding
  - Partition a vector into M sub-vectors and quantize each sub-vector using a separate codebook with K (256 by default) codewords
  - Store only the corresponding **codeword IDs,** each with $Mlog_2 K$ bits
- Approximate Distance Computation
  - Precompute a **lookup table (LUT)** for each subspace
    - query-to-centroid distance for every subspace and codeword
  - Compute the approximate distance with **table lookups + additions**
  - ADC is a biased distance estimator without theoretical error bound



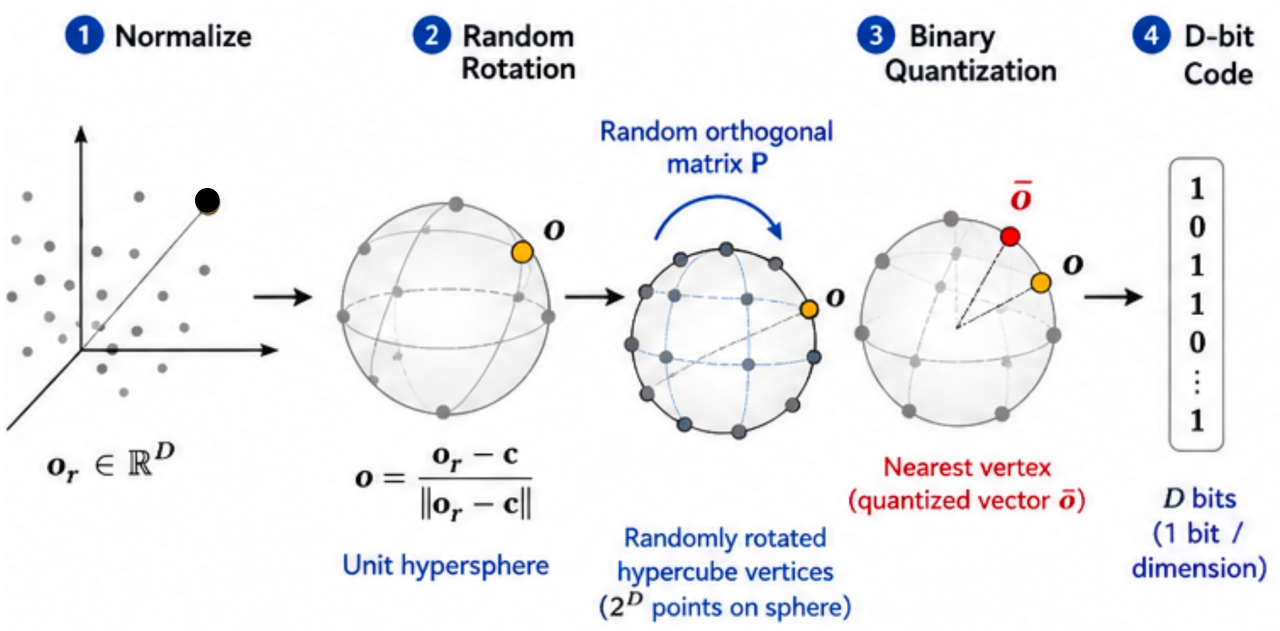
$$\hat{d}(\mathbf{q}, \mathbf{x}) = \sum_{i=1}^{M} LUT_i[c_i]$$

$$PQ(\mathbf{x}) = [c_1, c_2, \ldots, c_M]$$

[1] PQ: Jegou et al., Product quantization for nearest neighbor search. **IEEE TPAMI 2011**.
[2] The figure from: Le DD, et. al.. Efficient retrieval of matrix factorization-based top-k recommendations: A survey of recent approaches. **Journal of Artificial Intelligence Research**. 2021

Randomized Binary Quantization with Error Guarantees

- RaBitQ Pipeline
  - **Normalize** each vector $\mathbf{o}_r$ to **o** on a unit hypersphere and reduce distance estimation to inner-product estimation
  - Randomly **rotated** binary codebook $\mathcal{C} = \{-1/\sqrt{(D)}, +1/\sqrt{(D)}\}^D$ to obtain $C_{rand} = \{Px \mid x \in C\}$
  - Encode each vector **o** with *D* bits by $sign(P^{-1}\mathbf{o}) \rightarrow [+, -, +, +, -, \ldots +] \rightarrow 10110 \ldots 1$



[1] RaBitQ: Gao et al., RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. **SIGMOD 2024**.

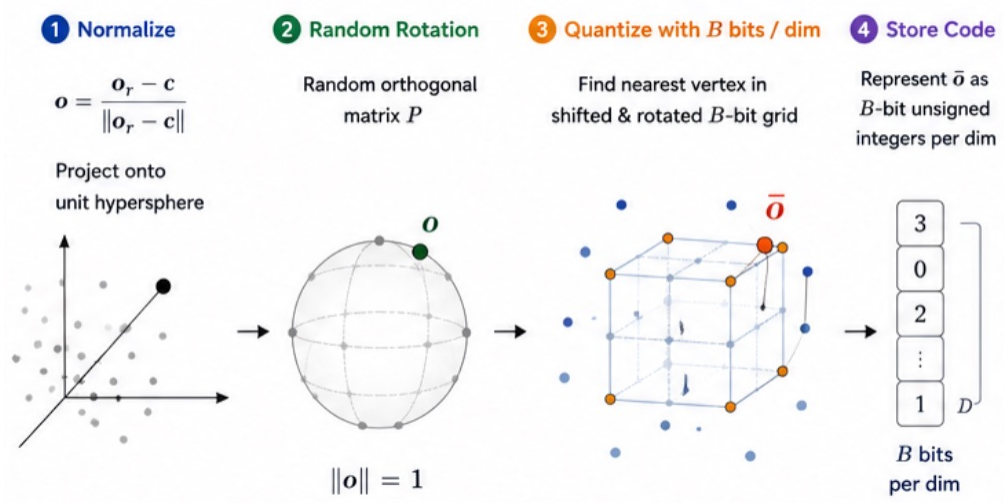
Randomized Binary Quantization with Error Guarantees

- From Distance to Inner Product after normalization
  - $\|o_r - q_r\|^2 = \|o_r - c\|^2 + \|q_r - c\|^2 - 2\|o_r - c\| \times \|q_r - c\| \times \langle \boldsymbol{o}, \boldsymbol{q} \rangle$
- Distance Estimation
  - Unbiased estimator: $E\left[\frac{\langle \bar{o}, q \rangle}{\langle \bar{o}, o \rangle}\right] = \langle o, q \rangle$ — $\langle \bar{o}, q \rangle$: computed in the query phase; $\langle \bar{o}, o \rangle$: pre-computed in the index phase
  - Sharp probabilistic error bound: $\left|\frac{\langle \bar{o}, \boldsymbol{q} \rangle}{\langle \bar{o}, \boldsymbol{o} \rangle} - \langle o, q \rangle\right| = O\left(\frac{1}{\sqrt{D}}\right)$ with high probability
- Efficient Distance Estimation
  - Transforms expensive floating-point distance computations into compact binary-integer inner products
  - Accelerate the computation by bitwise operations

[1] RaBitQ: Gao et al., RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. **SIGMOD 2024**.

RaBitQ+: From 1 Bit/Dim to B Bits/Dim

- RaBitQ: D-dim float 32 -> D bits 32x compression, which needs rerank for high recall
- RaBitQ+ has the same pipeline as RaBitQ, but with B-bit quantization per dimension
  - $\mathcal{C} = \{-\frac{2^B-1}{2} + u | u = 0,1,\cdots,2^B-1\}^D$ and $C_{rand} = \{P\frac{x}{\|x\|} \mid x \in C\}$
- Efficient Distance Estimation is the same as RabitQ
  - Difference: incremental estimation with B-bit codes
    - Most Significant Bit for coarse estimation, and remaining bits for refinement

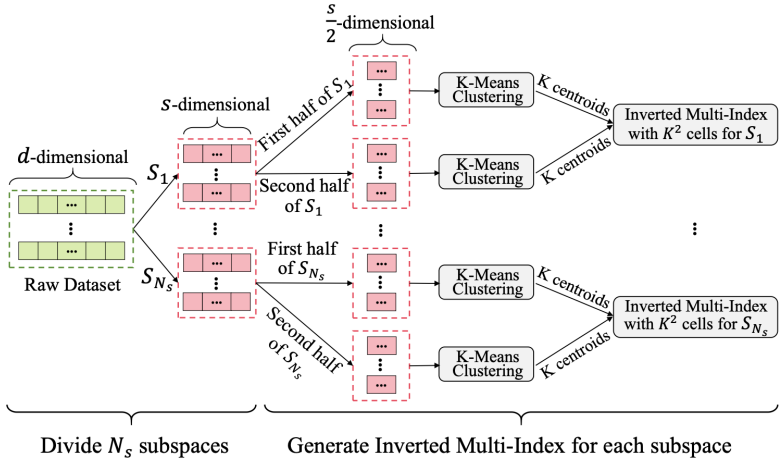


[1] RabitQ+: Gao et al., Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. **SIGMOD 2025**.

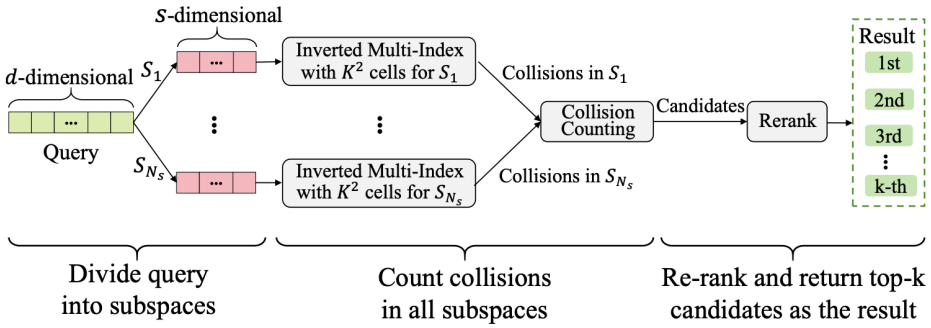
SuCo: A Collision-Based Proxy

- **Index Construction**
  - Like PQ, divide each vector into N_s subspaces
  - Cluster sub-vectors independently in each subspace and build an inverted multi-index for each subspace
- **Query Answering**
  - Project the query into the N_s subspaces
  - Compute SC-scores, i.e., the number subspaces where a data point collides with the query
    - x is one of the $(\alpha \cdot n)$-NNs of q in S_i
  - Vectors with higher SC-scores are more likely to be close to the query.

$$\|x-q\|^2 = \sum_{i=1}^{N_s} \|x^{(i)} - q^{(i)}\|^2 \quad \Rightarrow \quad SC(x,q) = \sum_{i=1}^{N_s} \mathbf{1}[x \text{ collides with } q \text{ in } S_i]$$



(a) Index construction.



(b) Query answering.

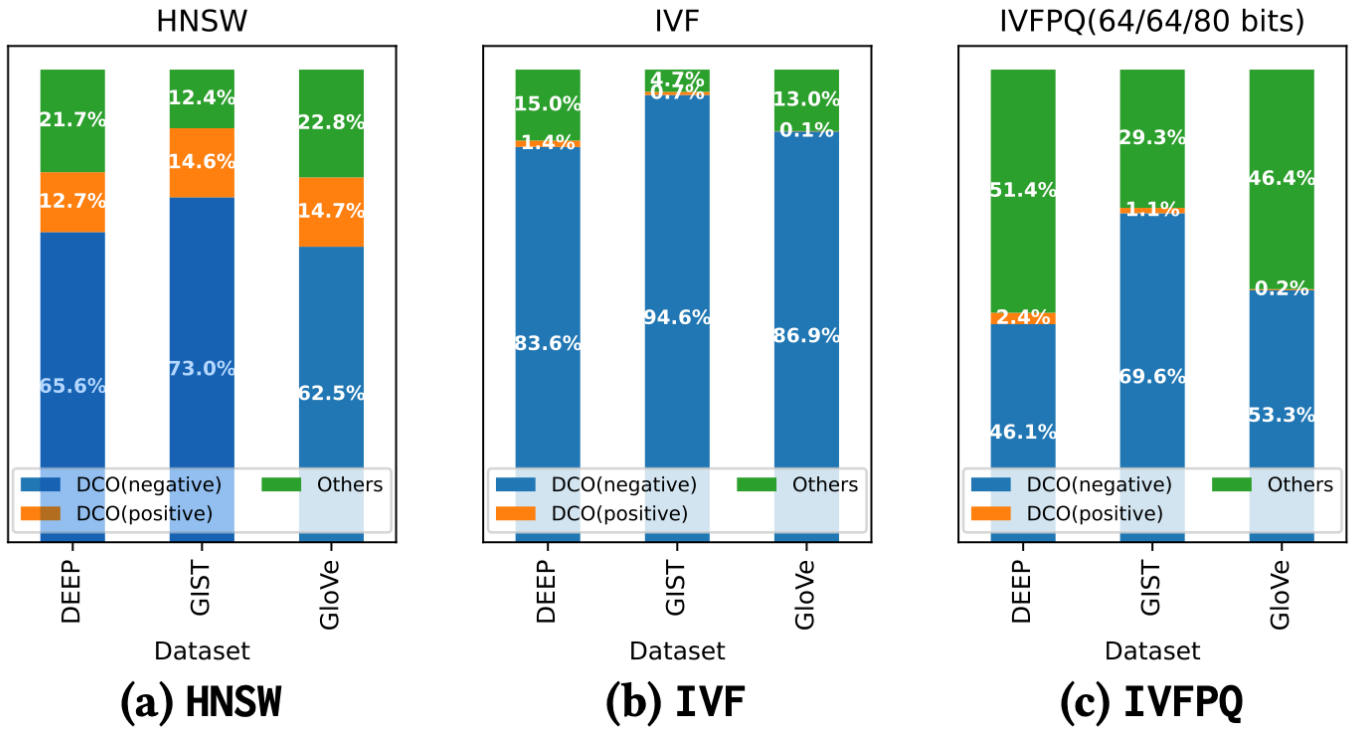
[1] SuCo: Wei et al., Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. **SIGMOD 2025**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
  - **Part 2.1**: Proximity Graph Based Approaches
  - **Part 2.2**: Quantization-Based Approaches
  - **Part 2.3**: Distance Computation Acceleration
  - **Part 2.4**: Hard and Out-of-Distribution Queries
  - **Part 2.5**: Secure Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Distance Computations in ANN Search

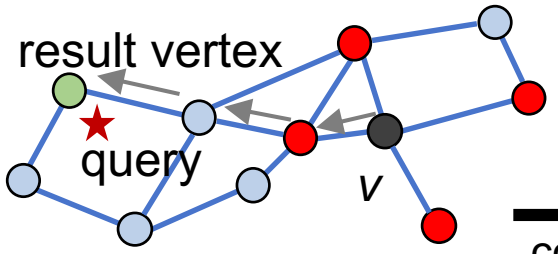
- Distance computations are the **main source of time overhead** in ANN search
- Speeding up distance calculation can improve the overall query performance



[1] Gao et al., High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. **SIGMOD 2023**.

Approximate Distance Estimation + Reranking

- Conduct ANNS on the index with quantization codes, e.g. PQ, RabitQ, RabitQ+ …
  - Each candidate is evaluated by the approximate distance, which requires less memory-access and CPU-computational cost
- Select several points with the smallest approximate distances to compute the real distance.



PQ/RabitQ

ANNS with compressed vectors

ID	Est. dist.
18	1.2
9	1.6
67	2.1

Rerank with original vectors

ID	Real dist.
9	1.32
18	1.54
67	1.86

[1] PQ: Jegou et al., Product quantization for nearest neighbor search. **PAIM 2010**.
[2] RaBitQ: Gao et al., RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. **SIGMOD 2024**.
[3] RabitQ (ext): Gao et al., Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. **SIGMOD 2025**.
[4] SymphonyQG: Gou et al., SymphonyQG: Towards Symphonious Integration of Quantization and Graph for Approximate Nearest Neighbor Search. **SIGMOD 2025**.

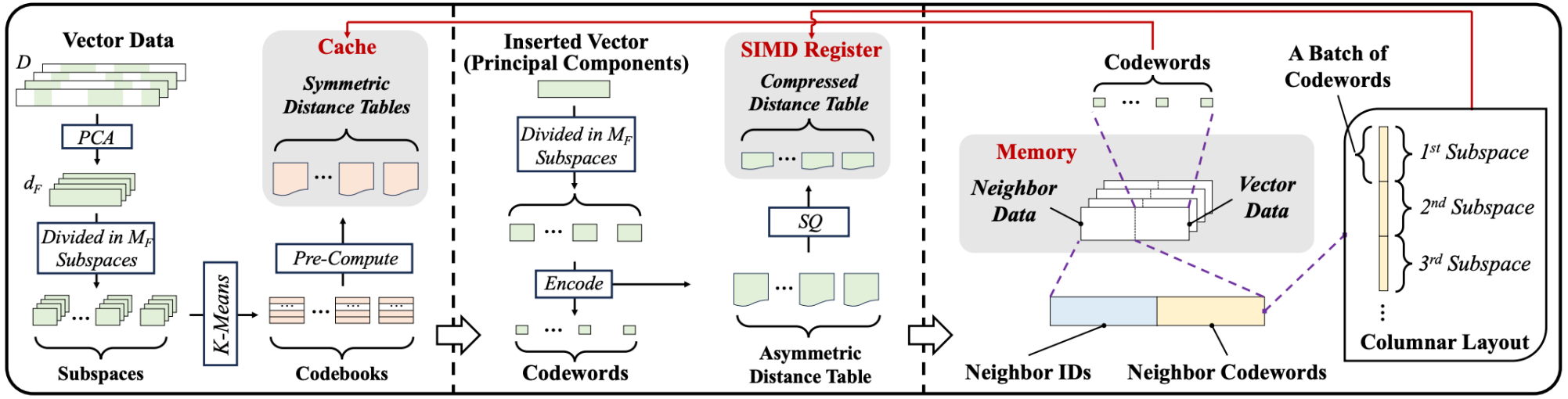
Lower Bound of Distances

- For each candidate u, first compute the lower bound. If the lower bound exceeds the distance of the current k-th best exact distance, u can be safely pruned; otherwise, compute its distance;
  - The lower bound should be tight enough and have low computational overhead.
- ADSampling first estimates the full distance by scaling the partial distance computed from randomly sampled dimensions, and then derive a probabilistic bound for early pruning
  - Randomizes the coordinate representation so that the distance contributions are distributed more evenly
- DDC exploits data-aware projection and correction to derive tighter distance bounds for more effective candidate pruning.
  - Random Projection → PCA / optimal orthogonal projection

[1] ADSampling: Gao et al., High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. **SIGMOD 2023**.
[2] DDC: Yang et al., Effective and General Distance Computation for Approximate Nearest Neighbor Search. **ICDE 2025**.

In-Register SIMD Distance Computation

- Flash accelerates HNSW index construction
  - HNSW index construction does not always need exact distance computations
- Compress vectors into compact representations that fit SIMD registers, enabling register-resident distance comparisons with fewer memory loads
  - Distance tables between codewords can be precomputed and stored in the cache.
  - The codewords belonging to the same subspace for many different neighbors are stored together.
    - Allows the CPU to load and process a batch of neighboring vectors simultaneously.



[1] Wang et al., Accelerating Graph Indexing for ANNS on Modern CPUs. **SIGMOD 2025**.

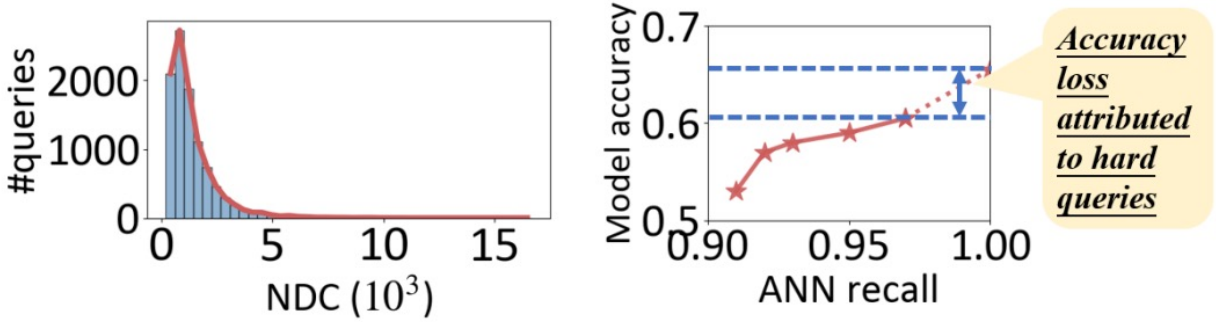
Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
  - **Part 2.1**: Proximity Graph Based Approaches
  - **Part 2.2**: Quantization-Based Approaches
  - **Part 2.3**: Distance Computation Acceleration
  - **Part 2.4**: Hard and Out-of-Distribution Queries
  - **Part 2.5**: Secure Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

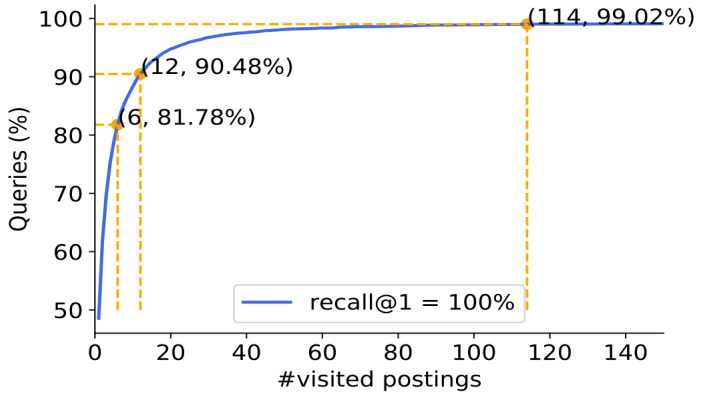
Query Hardness

- Different queries have **varied difficulty** in finding their top-k nearest neighbors
  - Evaluate the query hardness in the graph-based index.

- **How can we evaluate the hardness of an individual query?**



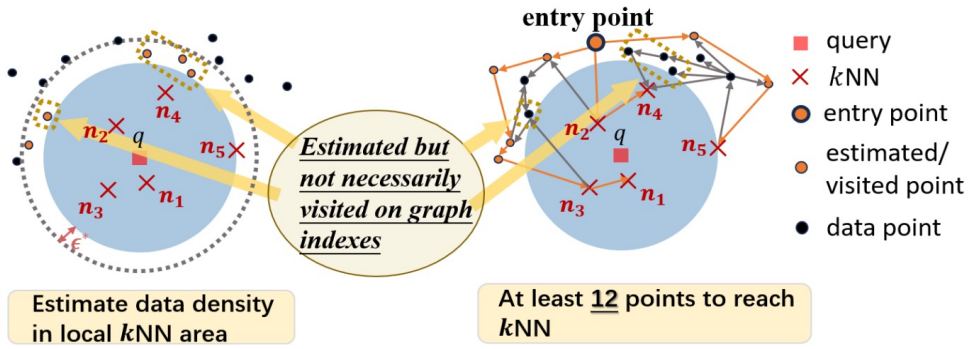
(a) Performance variance on Deep (b) Detriment to a RAG task



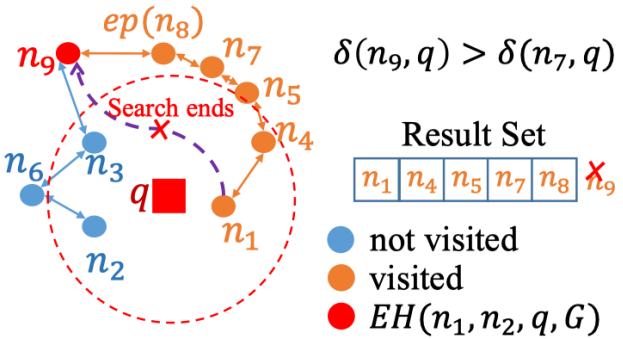
[1] Wang et al., ***Steiner***-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. **VLDB 2025**.
[2] Hua et al., Dynamically Detect and Fix Hardness for Efficient Approximate Nearest Neighbor Search. **SIGMOD 2026**.

Query Hardness

- Local Intrinsic Dimensionality LID $LID_q(r) = \lim_{\epsilon \to 0^+} \frac{ln(F((1+\epsilon)r)/F(r))}{ln(1+\epsilon)}$
  - F() is the cumulative distribution function of distances to query point q
- ***Steiner***-Hardness: models hardness from graph connectivity via Directed Steiner Tree
  - The measure estimates the minimum graph structure that must be traversed to connect the entry point to these nearest neighbors.
- Escape Hardness: models hardness from graph reachability
  - the smallest NN-rank radius needed before v becomes reachable from u



(a) LID (b) Minimum Effort

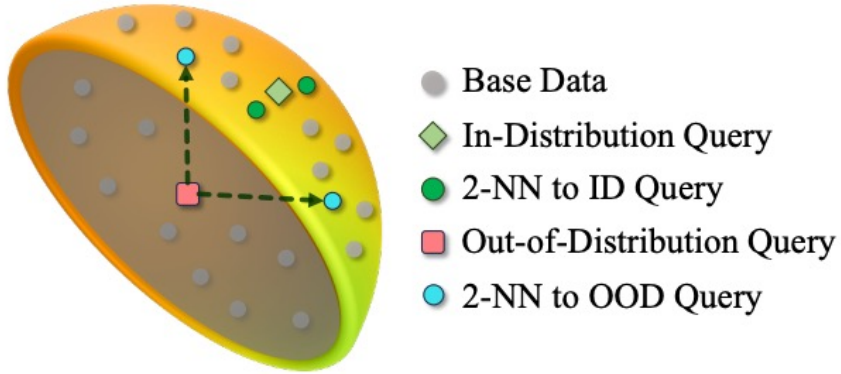


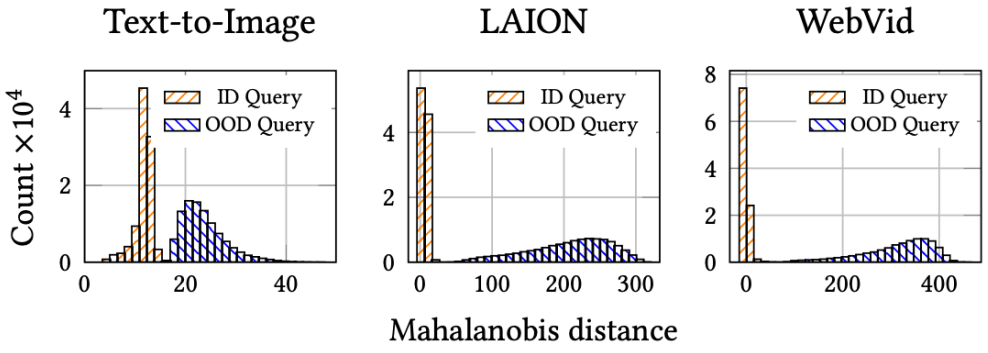
(a) An intuitive explanation of EH

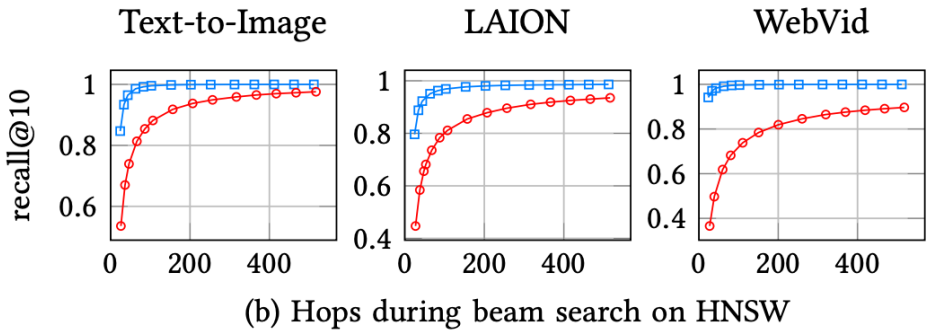
[1] Wang et al., ***Steiner***-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. **VLDB 2025**.
[2] Hua et al., Dynamically Detect and Fix Hardness for Efficient Approximate Nearest Neighbor Search. **SIGMOD 2026**.

Out-of-distribution Queries

- Cross-modal ANNS aims to use the data vector from one modality (e.g., texts) as the query to retrieve the most similar items from another (e.g., images or videos)
  - Jointly trained multimodal embedding aligns semantics but ensure the same embedding distributions or local-neighbor structures
  - Proximity graphs have poor performance on processing **out-of-distribution queries**.



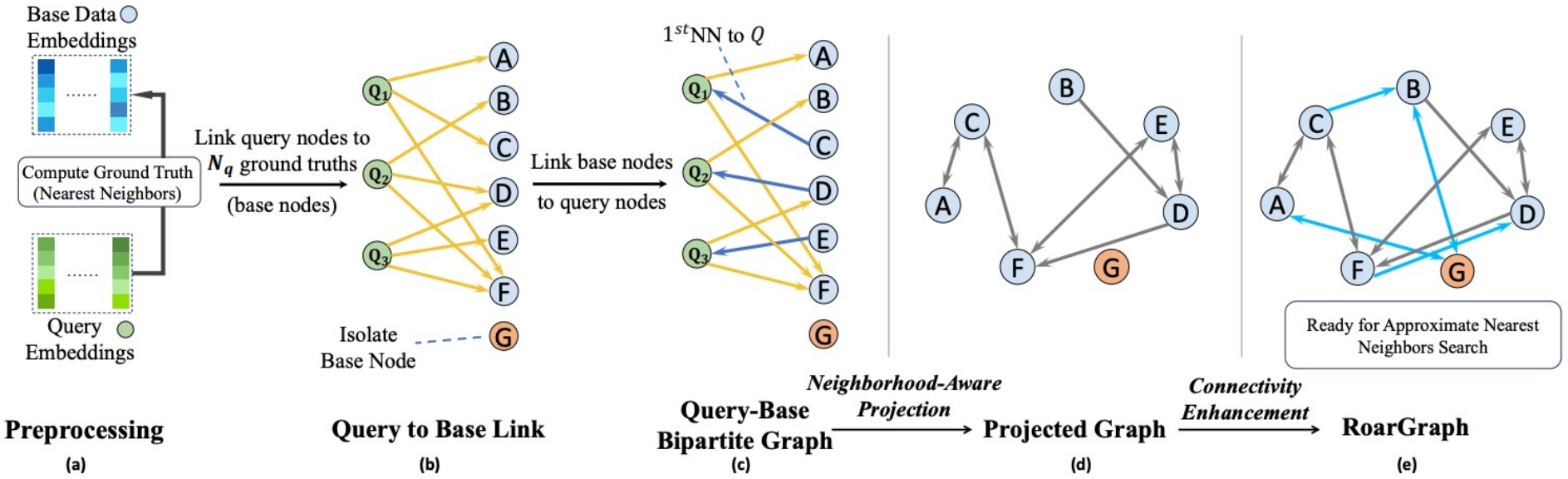




[1] Jaiswal et al., OOD-DiskANN: Efficient and Scalable Graph ANNS for Out-of-Distribution Queries. **ArXiv 2022**.
[2] Chen et al., RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. **VLDB 2024**.

Using OOD Queries during Index Construction

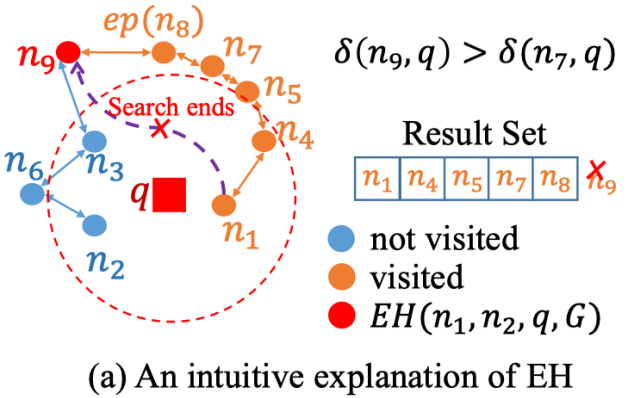
- RobustVamana incorporates historical OOD queries during index construction
- RoarGraph avoids permanent query-node insertion
  - Builds a bipartite graph between base vectors and historical OOD queries
    - These query nodes reveal relationships that are difficult to observe from the base vectors alone
  - Eliminates query nodes via a neighbor-aware projection



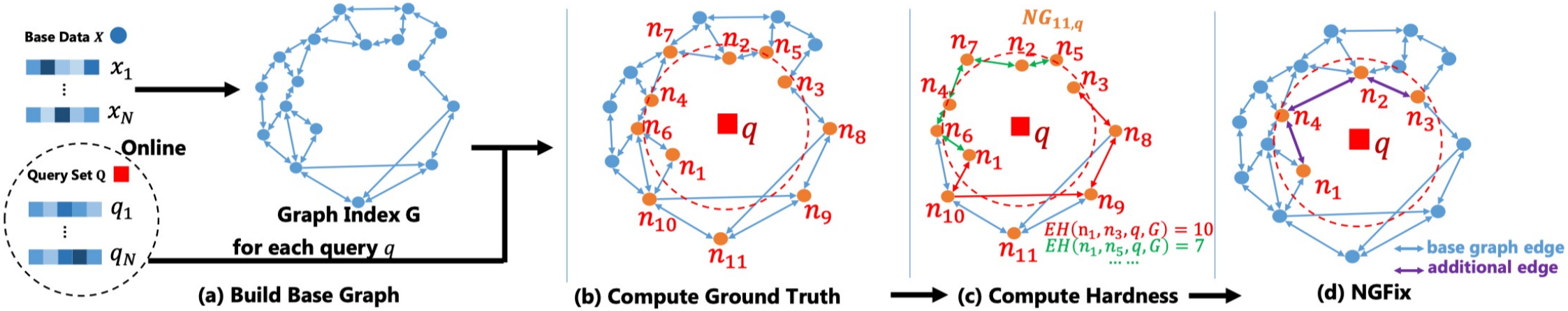
[1] Jaiswal et al., OOD-DiskANN: Efficient and Scalable Graph ANNS for Out-of-Distribution Queries. **ArXiv 2022**.
[2] Chen et al., RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. **VLDB 2024**.

Search-Time Graph Repairation

- NGFix repairs defective graph regions at search time based on Escape Hardness
  - RFix focuses on the first search stage: adds edges from the nearest point found so far to nodes closer to the query
  - For second search stage: repairs the graph around a historical query and greedily inserts edges to reduce the Escape Hardness



(a) An intuitive explanation of EH

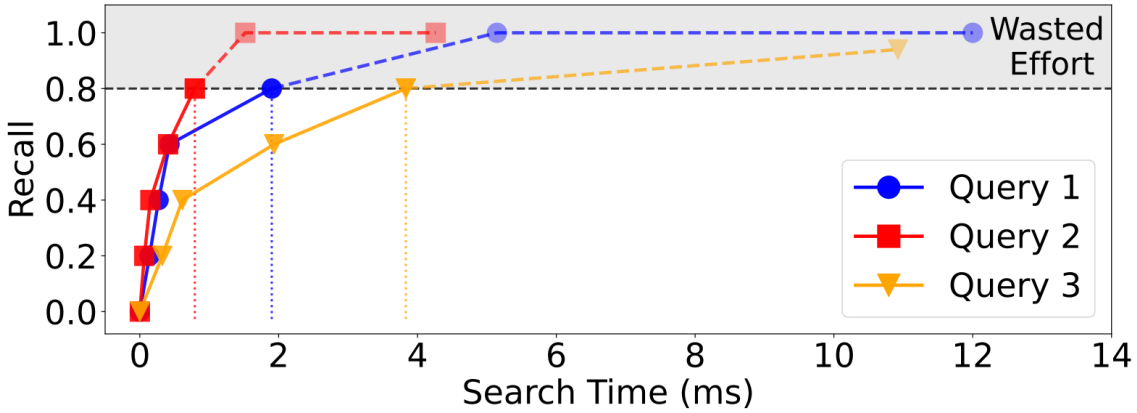


(a) Build Base Graph (b) Compute Ground Truth → (c) Compute Hardness → (d) NGFix

[1] Hua et al., Dynamically Detect and Fix Hardness for Efficient Approximate Nearest Neighbor Search. **SIGMOD 2026**.

Learning-based Approaches

- Different queries have **varying degrees of difficulty** in achieving a good recall.
- Different applications and different (classes of) users have **diverse requirements** for search quality.
- A fixed search budget cannot handle these differences efficiently.
  - If we use the same termination point for all three queries, we either terminate too early for the harder queries or waste effort on the easier ones.
- DARTH uses a learning-based method for providing target declarative recall on top of an ANNS index by employing an adaptive early termination strategy integrated into the search algorithm.



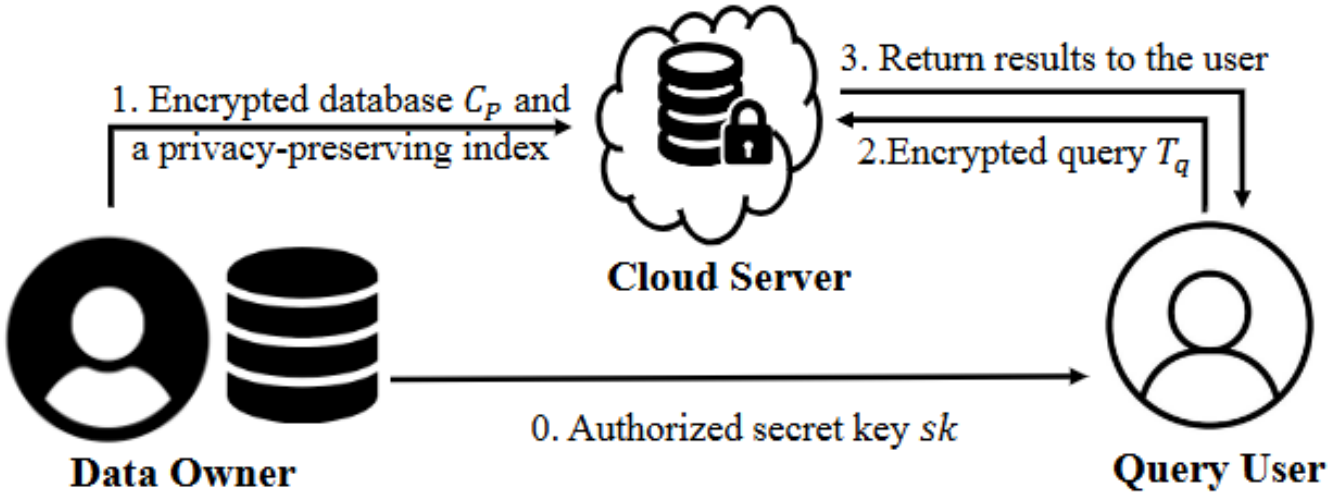
[1] Chatzakis et al. *DARTH: Declarative Recall Through Early Termination for Approximate Nearest Neighbor Search*. **SIGMOD 2026**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
  - **Part 2.1**: Proximity Graph Based Approaches
  - **Part 2.2**: Quantization-Based Approaches
  - **Part 2.3**: Distance Computation Acceleration
  - **Part 2.4**: Hard and Out-of-Distribution Queries
  - **Part 2.5**: Secure Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Secure Similarity Search

- Using cryptographic techniques to protect the privacy of the database vectors and query
- As widely recognized, a secure solution should satisfy three properties:
  - (P1) **Privacy-Preserving**: not recover sensitive vectors beyond the permitted leakage profile
  - (P2) **Efficient and Accurate**: remain fast enough and should preserve retrieval quality
  - (P3) **Minimizing User Involvement and Interaction**
- Key components: **vector encryption** and **secure index**



[1] Yingfan Liu, et. al.. Privacy-Preserving Approximate Nearest Neighbor Search on High-Dimensional Data. **ICDE 2025**.

Vector Encryption

- Vector encryption protects the privacy of vectors while enabling similarity search
- **Distance incomparable encryption**
  - Cannot answer whether or not $dist(p,q) < dist(o,q)$ using their encrypted vectors
  - E.g., classical encryption methods AES/DES
  - The user retrieves and decrypts candidate vectors locally for distance comparison
  - Leading to inefficiencies and under utilization of cloud resources
- **Distance comparable encryption**
  - The cloud directly answers the distance comparison using their encrypted vectors
  - Asymmetric scalar-product-preserving encryption (ASPE) → leaks information under relevant attack models
  - Distance-comparison-preserving encryption (DCPE) → inaccurate
  - Homomorphic encryption (HE) → high communication overhead
  - Symmetric matrix encryption (AME) → inefficient

[1] ASPE: Wong et al., Secure knn computation on encrypted databases. **SIGMOD 2009**.
[2] DCPE: Fuchsbauer et al., Approximate distance-comparison-preserving symmetric encryption. **SCN 2022**.
[3] HE: Guan et al., Toward oblivious location-based k-nearest neighbor query in smart cities. **IEEE Internet Things 2021**.
[4] AME: Zheng et al., Achieving practical and privacy-preserving knn query over encrypted data. **IEEE TDSC 2024**.

Vector Encryption

- Distance comparable encryption (DCE) is designed to reveal the results of distance comparisons without revealing the underlying vectors.
- Inspired by AME, DCE just reveals the results of distance comparison.
- Inspired by ASPE, DCE just involves the operations between vectors.
- DCE consists of two phases:
  - **Vector Randomization:** Generates a random vector to add randomness and scramble attackers
  - **Vector Transformation:** Transforms the output of the first phase to make it KPA secure for resisting known-plaintext attacks

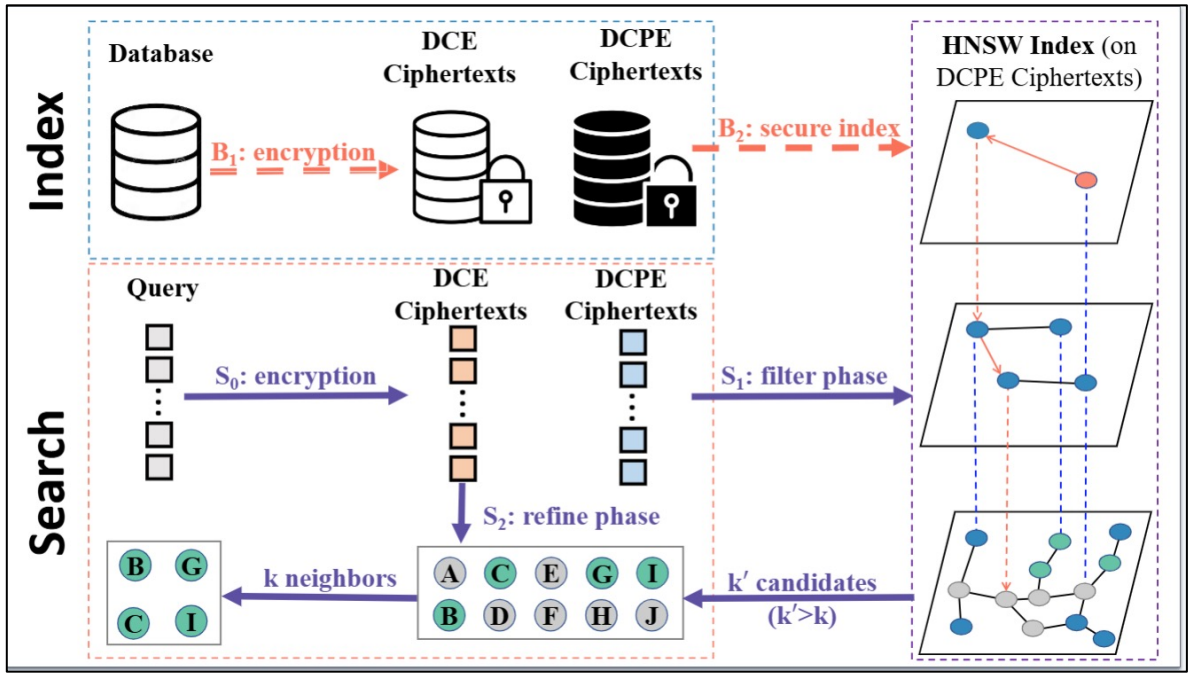
Encryption	Efficiency	Data Privacy	Non-Decryption Comparison	Ciphertext Computable
DCE	**High**	√	√	√
AME[1]	**Low**	√	√	√
ASPE[2]	**High**	×	√	√
HE[3]	**Low**	√	×	√
AES[4]	**Low**	√	×	×

"√" : denotes the scheme satisfies the objective.
"×" : denotes the scheme does not satisfy the objective.

[1] AME: Zheng et al., Achieving practical and privacy-preserving knn query over encrypted data. **IEEE TDSC 2024**.
[2] ASPE: Wong et al., Secure knn computation on encrypted databases. **SIGMOD 2009**.
[3] HE: Guan et al., Toward oblivious location-based k-nearest neighbor query in smart cities. **IEEE Internet Things 2021**.
[4] AES: Selent et al., Advanced encryption standard. **Rivier Academic Journal 2010**.
[5] DCE: Yingfan Liu, et. al.. Privacy-Preserving Approximate Nearest Neighbor Search on High-Dimensional Data. **ICDE 2025**.

Secure Index

- A secure index efficiently identifies promising candidates over encrypted vectors
  - LSH and proximity graphs have been modified as the secure index
- PP-ANNS employs HNSW as the index and takes a Filter-and-refine search strategy
- Filter: HNSW + DCPE
  - HNSW is the SOTA index for high dimensional data
  - DCPE sacrifices accuracy in exchange for efficiency
- Refine: Max heap + DCE
  - A max heap maintains the current top-k candidates
  - DCE compares distance exactly



[1] PP-ANNS: Yingfan Liu, et. al.. Privacy-Preserving Approximate Nearest Neighbor Search on High-Dimensional Data. **ICDE 2025**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
  - **Part 3.1**: Search on Multi-Vector Objects
  - **Part 3.2**: Search on Single-Vector Objects
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Multi-Similarity Search

- **Approximate Nearest Neighbor (ANN) Search with Multiple Vectors**
  - One object may contain multiple modularity-specific vector representations
  - **Example**: find the top 4 similar images that are related to both ${input_image} and "Happy Birthday".

Happy Birthday

```
SELECT poster_template
FROM posters
ORDER BY DISTANCE(image_mod, ${query_image})+
         DISTANCE(text_mod, "happy birthday")
LIMIT 4
```

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
  - **Part 3.1**: Search on Multi-Vector Objects
  - **Part 3.2**: Search on Single-Vector Objects
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Multi-Similarity Search on Multi-Vector Objects

- **One-to-One Vector Matching**
  - Each object $o_i \in O$ is represented as $o_i = (v_{i,1}, v_{i,2}, \cdots, v_{i,m})$, where $v_{i,1}, v_{i,2}, \cdots, v_{i,m}$ may lie in different embedding spaces and thus may have different dimensionalities
  - A query is similarly specified as $\mathrm{q} = (q_1, q_2, \cdots, q_m)$, and the query distance is defined by aggregating per-field distances between corresponding vector pairs $(v_{i,j}, q_j)$, e.g., $\sum_{j=1}^{m} w_j \delta(v_{i,j}, q_j)$.
  - A special case in this variant is **hybrid search**, where some of the vectors are a sparse vector, e.g., BM25-style term-weight vectors
    - Mostly zeros; excel at exact keyword matching, rare terminology, and precise lexical search
- **One-to-Many Vector Matching**
  - Each object $o_i \in O$ is represented as $o_i = \{v_{i,1}, v_{i,2}, \cdots, v_{i,m_i}\}$, where m_i may vary across objects, and each $v_{i,j} \in \mathbb{R}^d$ is an embedding.
    - E.g., token-level or patch-level multi-vector representation
  - A query is similarly specified as $\mathrm{q} = \{q_1, q_2, \cdots, q_t\}$, where $q_i \in \mathbb{R}^d$. The relevance is computed by fine-grained interactions between the two vector sets q and o_i, e.g., $\sum_{l=1}^{t} \max_{1 \le j \le m_i} \delta(v_{i,j}, q_l)$

ANN Search With Two Dense Vectors

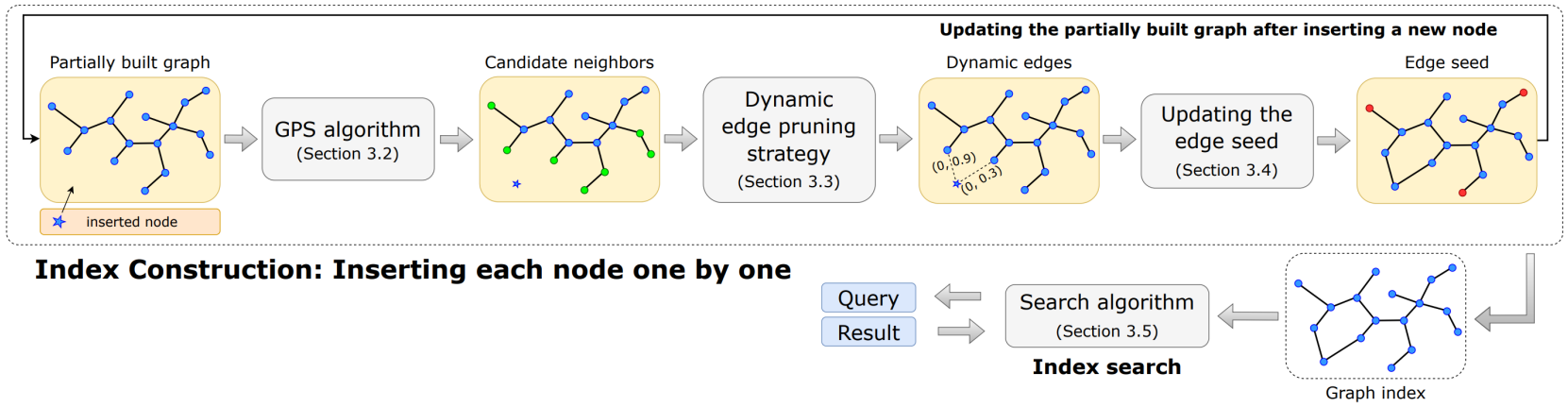
- **ANN search on two vectors** $Dist(q,o) = q.\alpha \times \delta_e(q,o) + (1 - q.\alpha) \times \delta_s(q,o)$

Hybrid Vector Query (HVQ): Given a dataset D, a hybrid vector query $q = \langle e, s, \alpha, k \rangle$ consists of two feature vectors $q.e \in E^d$ and $q.s \in E^m$, a hyperparameter $q.\alpha \in [0, 1]$, and the number of objects to be returned $q.k$. HVQ aims to retrieve k objects with the minimum distance $Dist(q, o)$:

$$Dist(q,o) = q.\alpha \times \delta_e(q,o) + (1 - q.\alpha) \times \delta_s(q,o), \quad (1)$$

where $\delta_e(q,o) = \frac{\delta(q.e,\, o.e)}{e_{max}}$ and $\delta_s(q,o) = \frac{\delta(q.s,\, o.s)}{s_{max}}$.

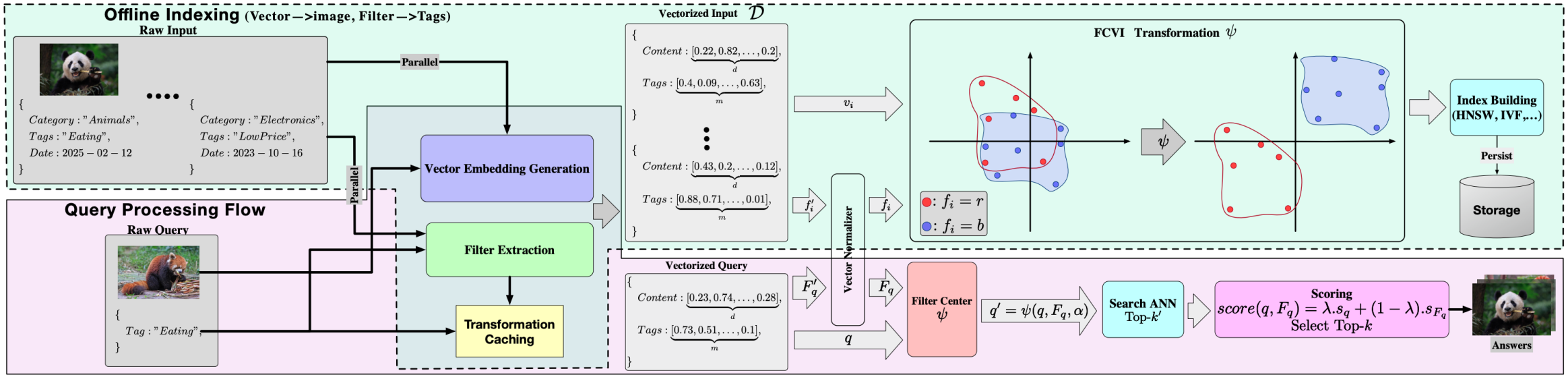
  - DEG
    - assigns each retained edge an active range, indicating the interval of weighted values for which the edge should be used



[1] DEG: Yin et al., DEG: Efficient Hybrid Vector Search Using the Dynamic Edge Navigation Graph. **SIGMOD 2025**.

Attribute-as-Vector Transformation

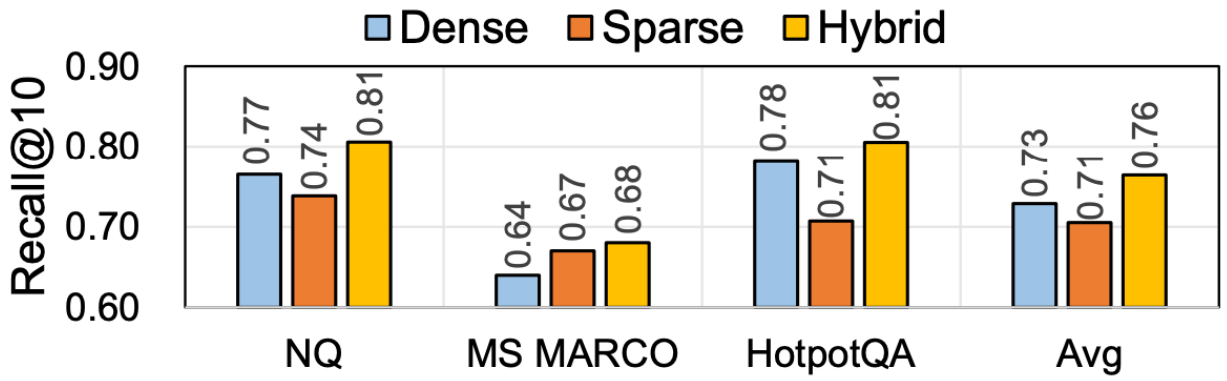
- **ANN search on two vectors**
  - FCVI represents attributes as a vector
    - An object $o_i \in O$ is represented as $o_i = (v_i, f_i)$, where f_i is filter vector.
      - For categorical attributes, use one-hot encoding or learned embeddings
      - For numerical attributes, normalize values to the range appropriate for transformation



[1] FCVI: Heidari et al., Filter-Centric Vector Indexing: Geometric Transformation for Efficient Filtered Vector Search. **Arxiv 2025**.

ANN Search With Dense and Sparse Vectors

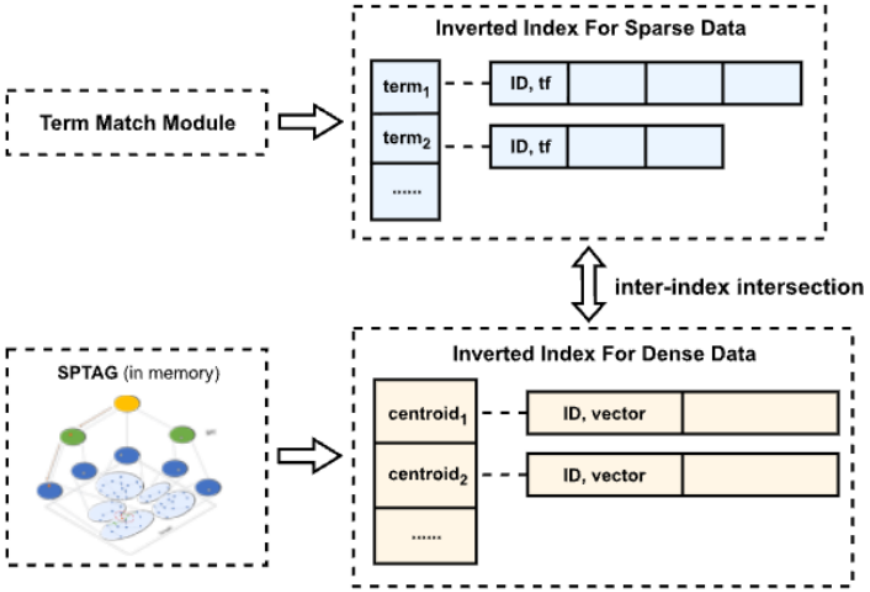
- Dense, such as Word2Vec and BERT-based DPR, learn low-dimensional dense vectors that **encode semantic and contextual information**.
  - For example, two passages may use different words but express similar meanings. Their dense representations can still be close to each other.
- Sparse, such as TF-IDF and BM25, rely on **keyword-frequency statistics** to map texts into high-dimensional sparse vectors.
  - Effective for exact keyword/term matching



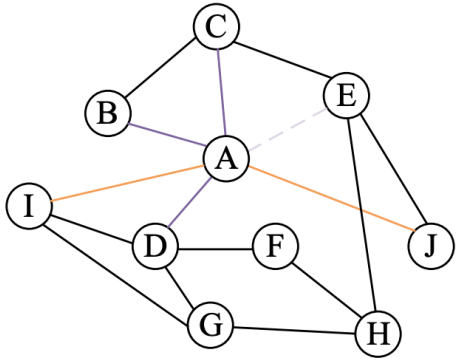
[1] Chen et al., OneSparse: A Unified System for Multi-index Vector Search. **WWW 2024**.
[2] Zhang et al., Efficient and Effective Retrieval of Dense-Sparse Hybrid Vectors using Graph-based Approximate Nearest Neighbor Search. **arXiv 2024**.
[3] Li et al., All-in-one Graph-based Indexing for Hybrid Search on GPUs. **VLDB 2026**.

Indexing Dense and Sparse Vectors

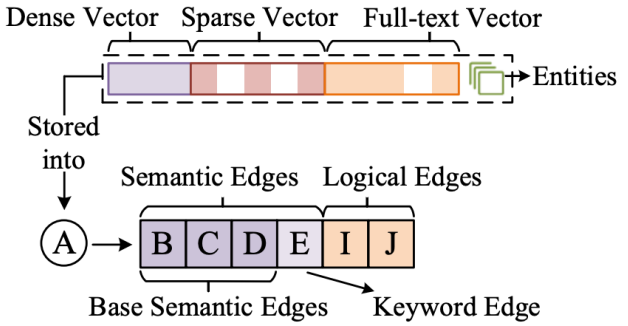
- To exploit the complementary strengths of both methods and mitigate their respective limitations, the dense-sparse hybrid method has been developed.
  - Multiple Indices [1]
  - Unified Graph Index [2][3]



OneSparse



(a) Structure of the hybrid index.



(b) Detailed contents of each node.

Allan-Poe

[1] Chen et al., OneSparse: A Unified System for Multi-index Vector Search. **WWW 2024**.
[2] Zhang et al., Efficient and Effective Retrieval of Dense-Sparse Hybrid Vectors using Graph-based Approximate Nearest Neighbor Search. **arXiv 2024**.
[3] Li et al., All-in-one Graph-based Indexing for Hybrid Search on GPUs. **Arxiv 2026**.

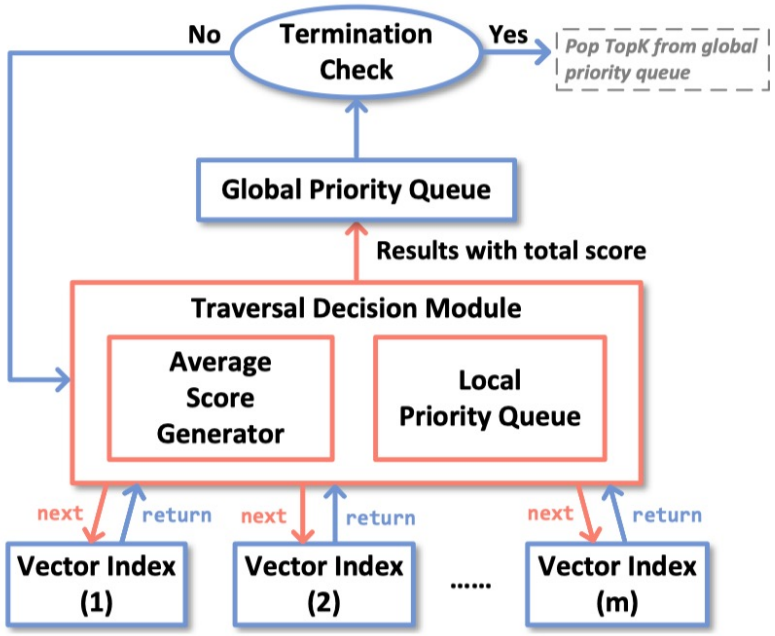
ANN Search With Multiple Vectors

- **Iterative Candidate Expansion**
  - Milvus
- **Multi-Index Traversal with Guidance**
  - VBASE

Algorithm 2: Iterative merging

```
1 k' ← k;
2 while k' < threshold do
      // run top-k' processing for each q.v_i on D_i
3     foreach i do
4         R_i ← VectorQuery(q.v_i, D_i, k');
5     if k results are fully determined with NRA [19] on all R_i
      then
6         return top-k results;
7     else
8         k' ← k' × 2;
9 return top-k results from ∪_i R_i;
```

Milvus



VBASE

[1] Milvus: Wang et al., Milvus: A Purpose-Built Vector Data Management System. **SIGMOD 2021**.
[2] VBASE: Zhang et al., VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. **OSDI 2023**.

Multi-Dense One-to-Many Vector Search

- Convert this problem into ANN-based searches by transforming into a single vector
  - MUVERA constructs a fixed-dimensional encoding for a variable-sized vector set
- Optimizing late-interaction search
  - PLAID introduces cheaper centroid interaction and pruning to filter low-quality candidates early
  - ColBERTv2 reduces the storage footprint of token-level representations through aggressive residual compression, since one object may contain dozens or even hundreds of token vectors.
  - EMVB combines bit-vector pre-filtering with product quantization to accelerate late interaction via cheaply remove unlikely candidates, and reduce memory usage
- Utilize proximity graph
  - IGP builds a proximity graph for MaxSim-style retrieval and uses incremental greedy probing to obtain high-quality candidates with fewer probes
  - GEM proposes a native graph-based index by assigning each vector set only to the clusters of its most informative vectors, and builds a dual graph with intra-cluster graphs for local neighbor structure and a global graph for cross-cluster navigation

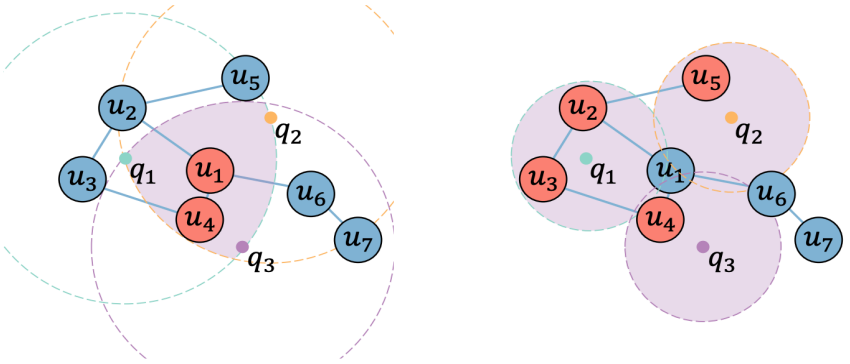
[1] MUVERA: Dhulipala et al., MUVERA: Multi-Vector Retrieval via Fixed Dimensional Encodings. **Arxiv 2024**.
[2] PLAID: Santhanam et al., PLAID: An Efficient Engine for Late Interaction Retrieval. **CIKM 2022**.
[3] ColBERTv2: Santhanam et al., ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. **NAACL 2022**.
[4] EMVB: Nardini et al., Efficient Multi-vector Dense Retrieval with Bit Vectors. **ECIR 2024**.
[5] IGP: Bian et al., IGP: Efficient Multi-Vector Retrieval via Proximity Graph Index. **SIGIR 2025**.
[6] GEM: Tian et al., GEM: A Native Graph-based Index for Multi-Vector Retrieval. **SIGMOD 2026**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
  - **Part 3.1**: Search on Multi-Vector Objects
  - **Part 3.2**: Search on Single-Vector Objects
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Multi-Similarity Search on Single-Vector Objects

- **All-k NN**: Given a dataset D and a query vector $q = [q_1, \cdots, q_m]$, the set of all-k NN of q is $R^{\forall}_{r^*} = \bigcap_{i=1}^{m}\{u \in D | \delta(u, q_i) \leq r^*\}$, where $r^* = \arg\min_r \ s.t. \ |\bigcap_{i=1}^{m}\{u \in D | \delta(u, q_i) \leq r\}| \geq k$.
  - Ranking vectors in D with all-radius: $r^*(u) = \max_{i \in \{1,\cdots,m\}} \delta(u, q_i)$ for a vector $u \in D$.
- **Any-k NN**: Given a dataset D and a query vector $q = [q_1, \cdots, q_m]$, the set of any-k NN of q is $R^{\exists}_{r^*} = \bigcup_{i=1}^{m}\{u \in D | \delta(u, q_i) \leq r^*\}$, where $r^* = \arg\min_r \ s.t. \ |\bigcup_{i=1}^{m}\{u \in D | \delta(u, q_i) \leq r\}| \geq k$.
  - Ranking vectors in D with any-radius: $r^*(u) = \min_{i \in \{1,\cdots,m\}} \delta(u, q_i)$ for a vector $u \in D$.

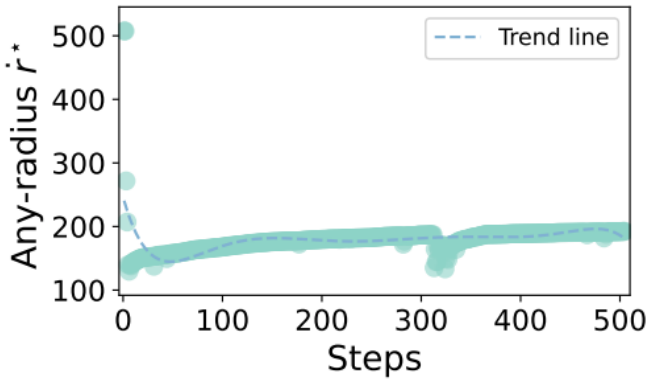


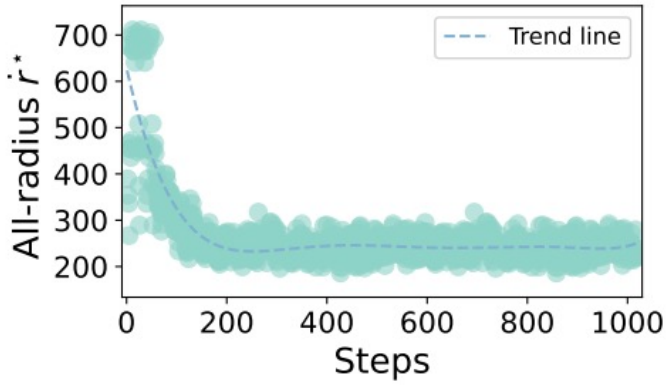
(a) All-k NN (b) Any-k NN

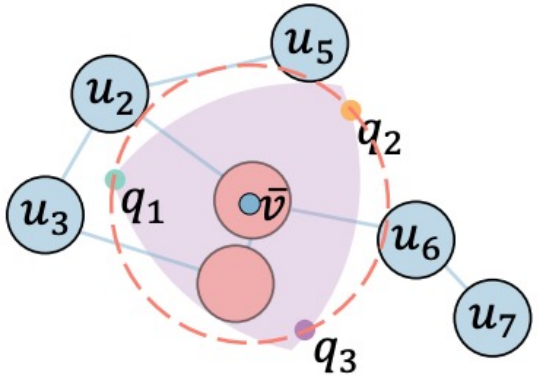
[1] Xie et al., Breaking the Single-Reference-Vector Barrier in Approximate Nearest Neighbor Search. **WWW 2026**.

Two Steps of Beam Search on Any/All-Radius

- **Non-continuous Space** of Any-k NN
  - Some of the balls centered at query vectors may not intersect with other balls.
  - First conduct separate 1-ANN searches for each query vector to approach the distinct m regions
  - Next, starting at the m 1-ANN of each query vector when applying beam search
- All-1 NN in the space is the center of the **minimum enclosing ball** of all query vectors.
  - First determines the optimal all-1 NN $\bar{v}$ in the space
  - Employs a beam search to find 1-ANN of $\bar{v}$
  - Start beam search at 1-ANN for the final results







[1] Xie et al., Breaking the Single-Reference-Vector Barrier in Approximate Nearest Neighbor Search. **WWW 2026**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
  - **Part 4.1**: Universal Index Approaches
  - **Part 4.2**: Dedicated Index Approaches
    - **Part 4.2.1**: Categorical Attributes
    - **Part 4.2.2**: Numerical Attributes
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Filtered Similarity Search

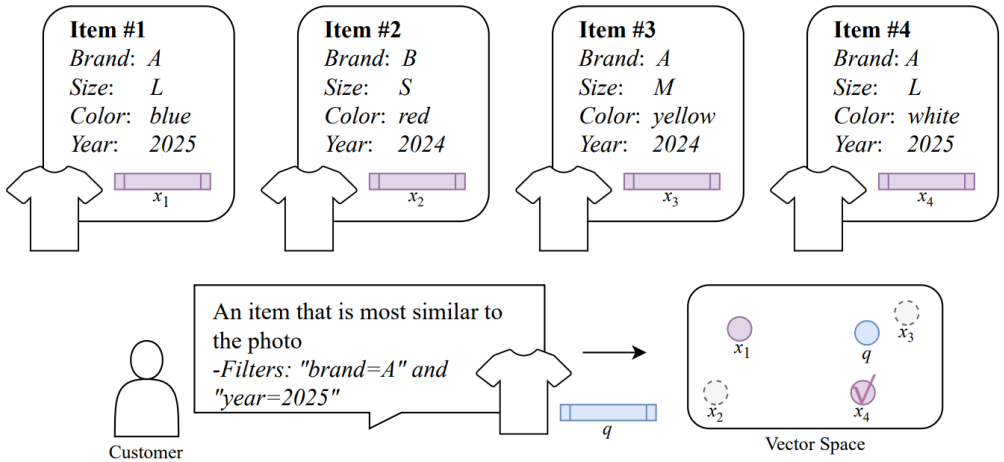
- **Approximate Nearest Neighbor (ANN) Search with Filters**
  - Similarity search on such objects/texts is to similarity search on vectors whose attributes satisfy the query predicates in the vector database
  - **Example**: find the top 4 similar coats to the ${input_image} priced below $200.

${input_image}
price < $200

```
SELECT id
FROM clothes_store
WHERE price < 200
ORDER BY DISTANCE(image, ${input_image})
LIMIT 4
```

Filtered Similarity Search

- Filtered similarity search assumes that each object stored in a vector database is associated not only with a vector representation, but also with (one or more) attributes.
  - Each object $o_i \in O$ is represented as a pair $o_i = (v_i, a_i)$, where $v_i \in \mathbb{R}^d$ is a vector representation, and a_i denotes its attribute(s).
  - A query contains a query vector $q_i \in \mathbb{R}^d$ and an attribute c_q with predicate p_q on it.
- **Categorical Attributes**
  - Let A_C denote a categorical domain. For each object $o_i = (v_i, a_i)$, the attribute is a set of labels $a_i = \{l_1, \cdots, l_{|a_i|}\}$, where $l_j \in A_C$.
  - $p_q(a_i) = [c_q \subseteq a_i]$, where $c_q \subseteq A_C$ is the query constrain.



Filtered Similarity Search

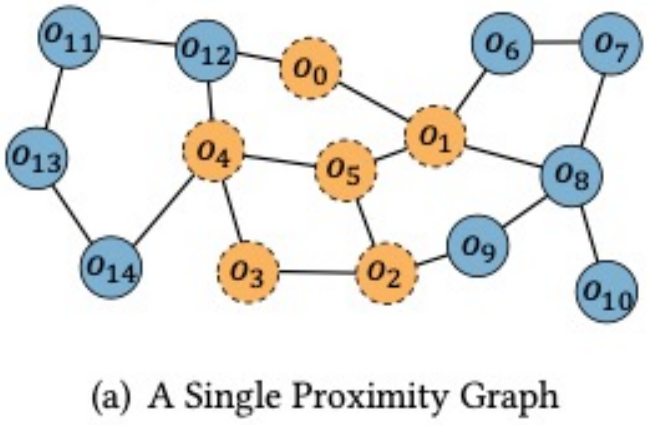
- Filtered similarity search assumes that each object stored in a vector database is associated not only with a vector representation, but also with (one or more) attributes.
  - Each object $o_i \in O$ is represented as a pair $o_i = (v_i, a_i)$, where $v_i \in \mathbb{R}^d$ is a vector representation, and a_i denotes its attribute(s).
  - A query contains a query vector $q_i \in \mathbb{R}^d$ and an attribute c_q with predicate p_q on it.
- **Numerical Attributes**
  - Let A_N denote a numerical domain. For each object $o_i = (v_i, a_i)$, the attribute a_i can be either a single value $a_i \in A_N$ or a closed interval $a_i = [l_i, r_i] \subseteq A_N$.
  - **Range-Filtered ANN**: $a_i \in A_N$
    - $p_q(a_i) = [a_i \in c_q]$, where $c_q \subseteq A_N$ is a query numerical range.
  - **Interval-Filtered ANN**: $a_i \subseteq A_N$
    - $p_q(a_i) = [a_i \subseteq c_q]$, where $c_q \subseteq A_N$ is a query numerical range.
  - **Timestamp-Filtered ANN**: $a_i \subseteq A_N$ is a time interval
    - $p_q(a_i) = [c_q \in a_i]$, where $c_q \in A_N$ is a query timestamp.

Tutorial Overview

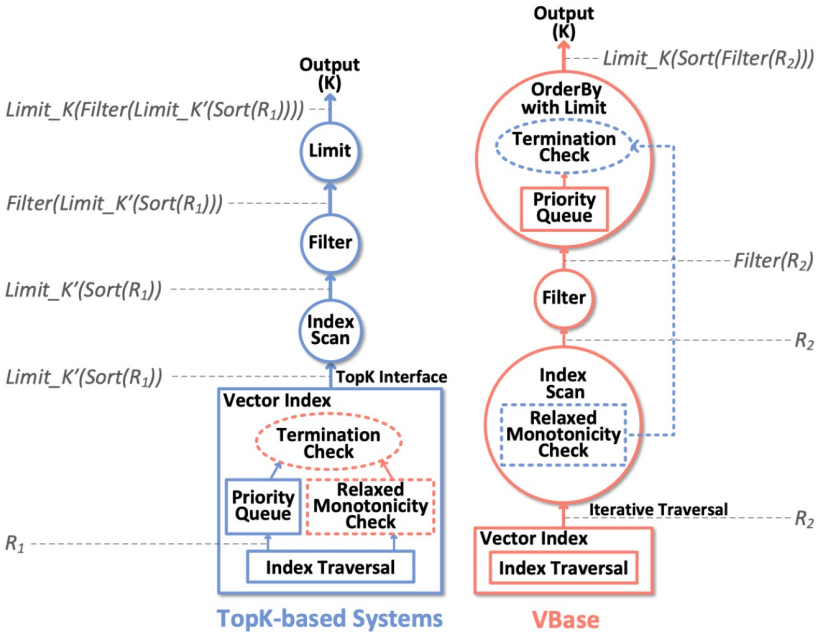
- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
  - **Part 4.1**: Universal Index Approaches
  - **Part 4.2**: Dedicated Index Approaches
    - **Part 4.2.1**: Categorical Attributes
    - **Part 4.2.2**: Numerical Attributes
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Universal Index: Search Algorithms

- It can support a query $q = (v_q, c_q)$ when $c_q = \emptyset$ **efficiently**.
- To support a query $q = (v_q, c_q)$ for $c_q \neq \emptyset$ over the single proximity graph built
  - *AnalyticDB-V* and *Milvus* have two strategies: pre-filtering and post-filtering.
  - *VBASE* unfolds in two distinct phases to search by v_q and search by c_q.



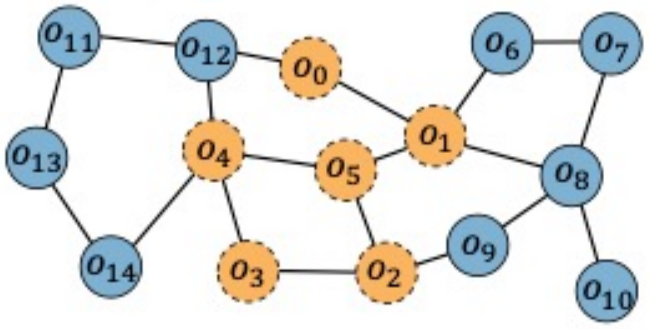
(a) A Single Proximity Graph



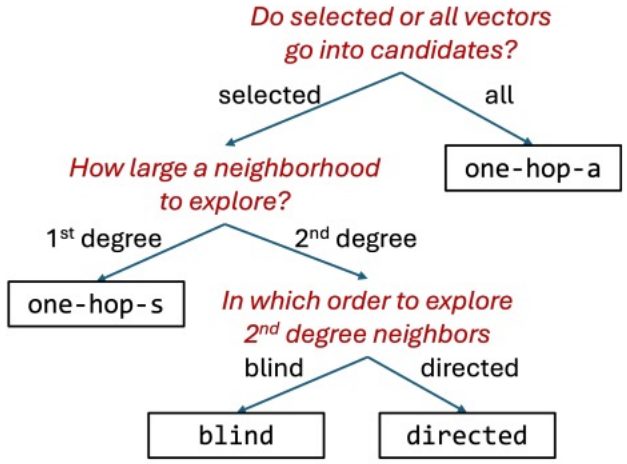
[1] ADBV: Wei et al., AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data. **VLDB 2020**.
[2] Milvus: Wang et al., Milvus: A Purpose-Built Vector Data Management System. **SIGMOD 2021**.
[3] VBASE: Zhang et al., VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. **OSDI 2023**.

Universal Index: Index Augmentation

- It can support a query $q = (v_q, c_q)$ when $c_q = \emptyset$ **efficiently**.
- To support a query $q = (v_q, c_q)$ for $c_q \neq \emptyset$ over the single proximity graph built
  - *ACORN* builds a new proximity graph by considering 2-hop neighbors during the graph construction and search phase, respectively.
  - *NaviX* proposes an adaptive 2-hop exploration strategy
    - Blind: continues to explore 2-hop neighbors in the default scanning order of 1-hop neighbors
    - Directed: explores 2-hop neighbors but first orders the 1-hop neighbors by their distances to the query vector v_q, biasing the search toward regions closer to v_q.



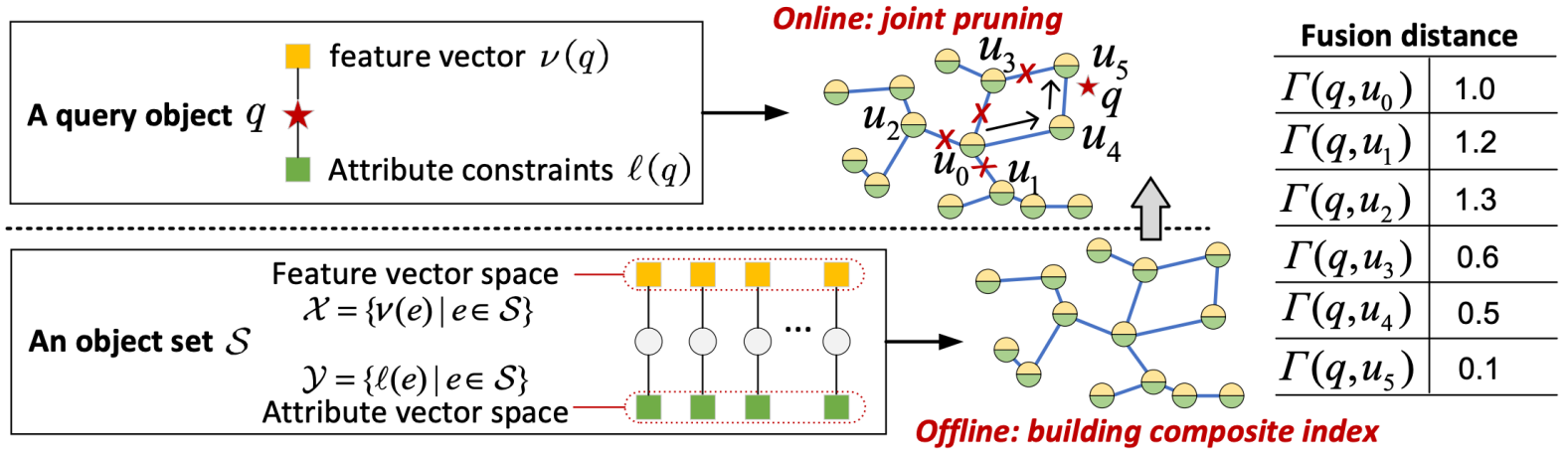
(a) A Single Proximity Graph



[1] ACORN: Patel et al., ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data.. **SIGMOD 2024**.
[2] NaviX: Sehgal et al., NaviX: A Native Vector Index Design for Graph DBMSs With Robust Predicate-Agnostic Search Performance. **VLDB 2025**.

Universal Index: Composite Index

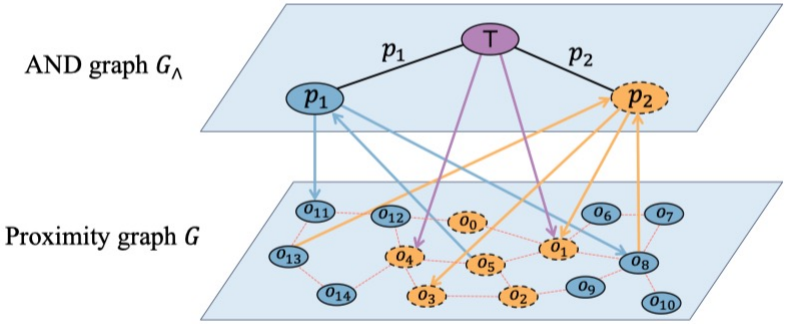
- *NHQ* builds a composite index and performs joint pruning during traversal
  - beam expansion considers both vector proximity to v_q and feasibility under c_q to prune unpromising branches
- *JAG* builds a graph that jointly captures vector proximity and attribute proximity
  - introduces attribute and filter distances, turning a binary constraint into a continuous navigational signal

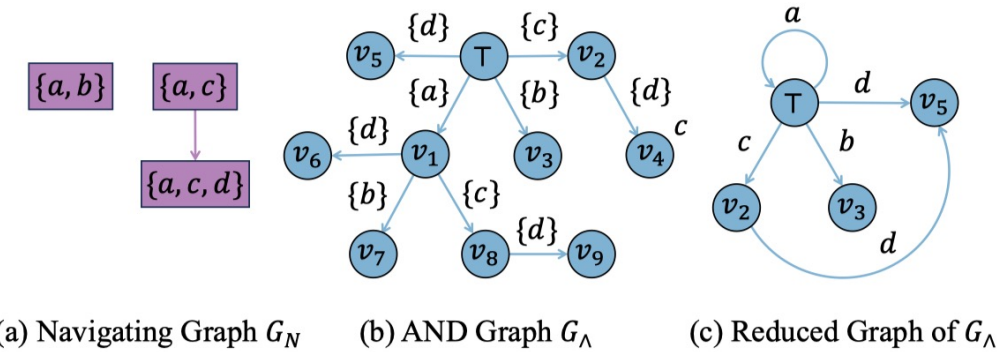


[1] NHQ: Navigable Proximity Graph-Driven Native Hybrid Queries with Structured and Unstructured Constraints. **Arxiv 2022**.
[2] JAG: Xu et al., JAG: Joint Attribute Graphs for Filtered Nearest Neighbor Search. **Arxiv 2026**.

Universal Index: Index Introduction

- *UniFilter* proposes **an enhanced proximity graph**
  - AND/OR graph + Proximity graph + Cross-edges
- **AND/OR Graph**
  - A rooted DAG with edge labels for objects in O with categorical/numerical attributes.
  - Each edge denotes an OR/AND semantics.
- **The Cross-Edges**: The edges between AND/OR graph and proximity graph
  - To minimize the maximum hop distance among valid objects
  - When invalid vector nodes are reached, the search can jump to regions containing valid objects.





(a) Navigating Graph G_N (b) AND Graph $G_\wedge$ (c) Reduced Graph of $G_\wedge$

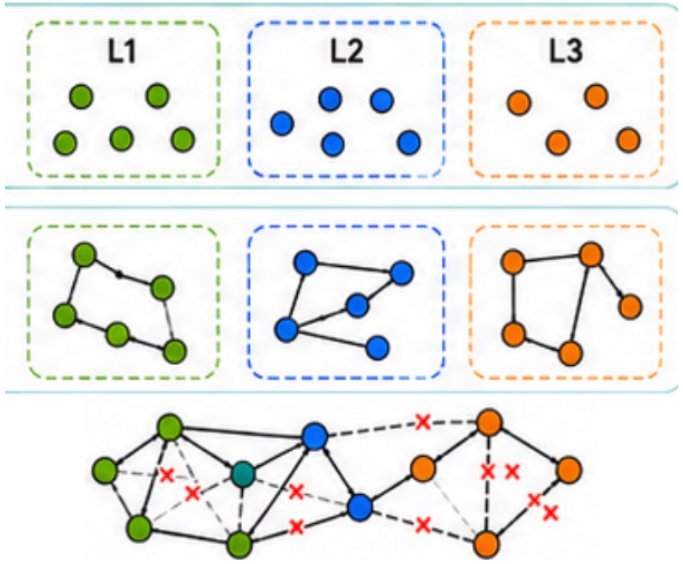
[1] Xie et al., Beyond Vector Search: Querying With and Without Predicates. **SIGMOD 2026**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
  - **Part 4.1**: Universal Index Approaches
  - **Part 4.2**: Dedicated Index Approaches
    - **Part 4.2.1**: Categorical Attributes
    - **Part 4.2.2**: Numerical Attributes
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Label-Aware PG Construction

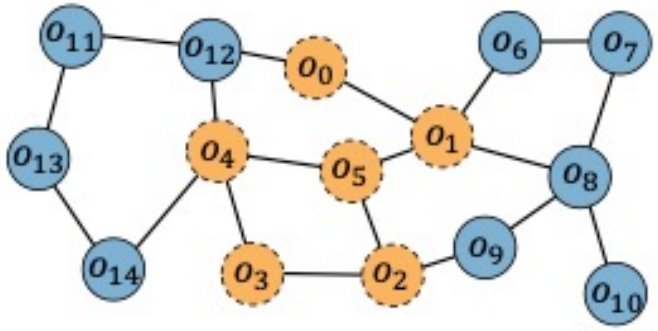
- **Support with relaxed pruning strategy**
  - FilteredVamana incrementally inserts vectors and biases neighbor discovery and edge selection toward vectors with compatible labels, so that predicate-satisfying nodes remain well connected in the resulting graph
  - StitchedVamana first builds a separate Vamana graph for each label filter and then merges these graphs into a single index by stitching their edges together, followed by pruning to bound the out-degree



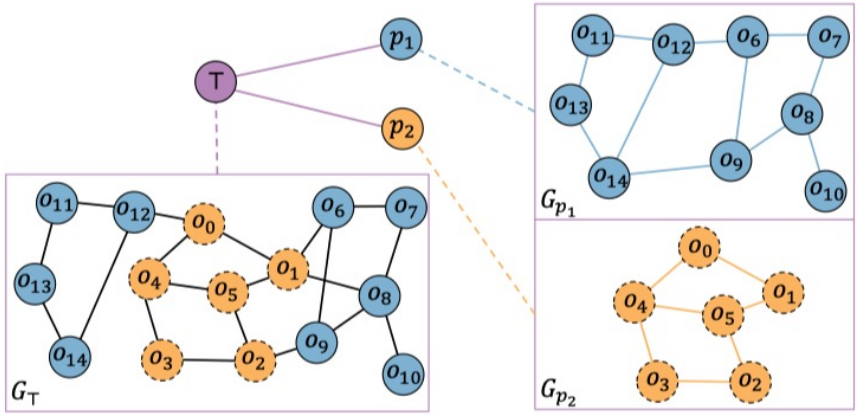
[1] FilteredDiskANN: Gollapudi et al., Filtered−DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. **WWW 2023**.

Dedicated Index Approaches

- Dedicated indexes are always **multi-proximity** graph approaches
  - The ideal case is to have a proximity graph G_{c_q} for a specific predicate c_q.
  - Reduce the number of materialized per-predicate indexes because construction and space costs are high.



(a) A Single Proximity Graph



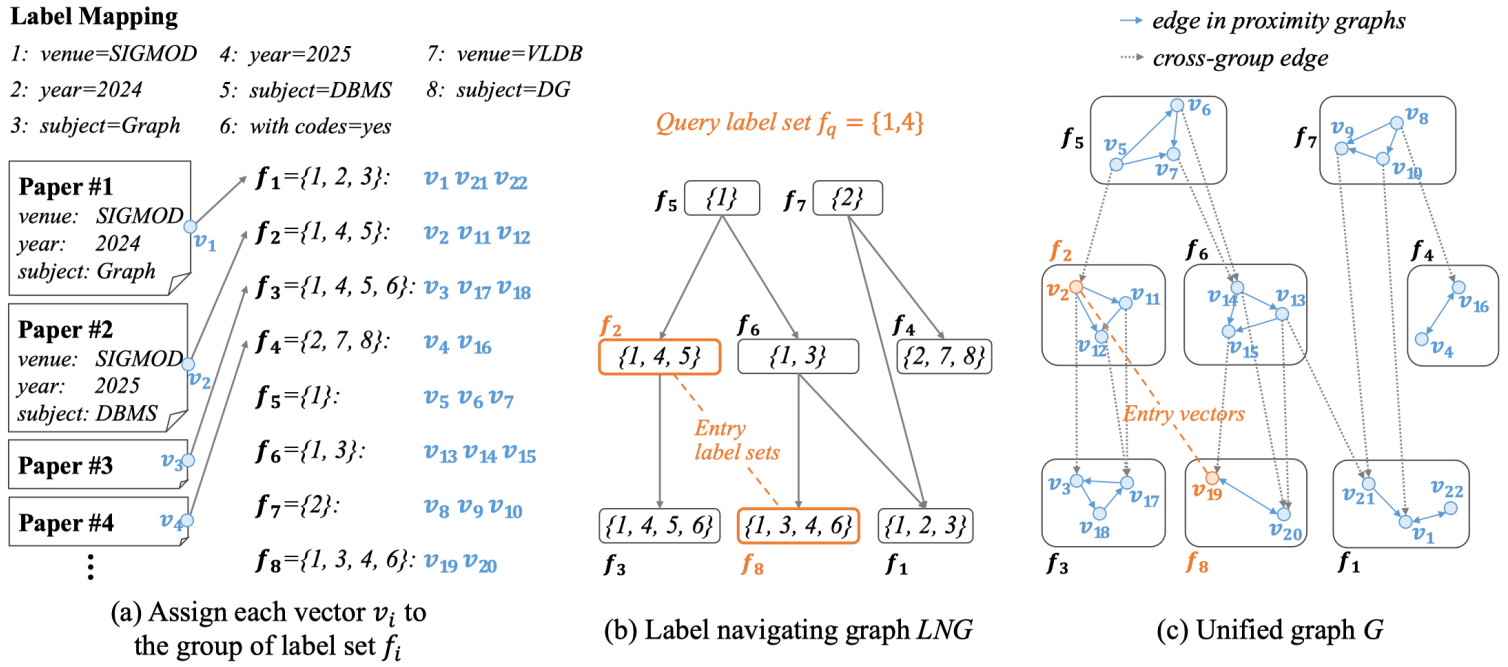
(b) Multi-Proximity Graphs

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
  - **Part 4.1**: Universal Index Approaches
  - **Part 4.2**: Dedicated Index Approaches
    - **Part 4.2.1**: Categorical Attributes
    - **Part 4.2.2**: Numerical Attributes
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Multiple PGs with a Navigating Graph

- UNG builds a PG for each distinct label set appearing in the dataset
  - It builds a label navigating graph over these label sets
    - For each query constrains c_q, it identifies the minimal supersets of c_q in label navigating graph, then searches the corresponding PGs



(a) Assign each vector v_i to the group of label set f_i

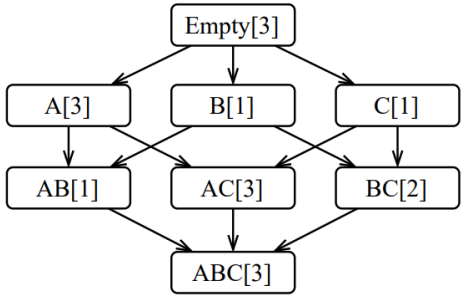
(b) Label navigating graph *LNG*

(c) Unified graph *G*

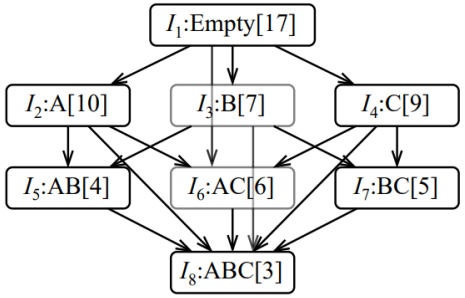
[1] UNG: Cai et al., Navigating Labels and Vectors: A Unified Approach to Filtered Approximate Nearest Neighbor Search. **SIGMOD 2025**.

Multiple PGs with a Navigating Graph

- ELI selectively indexes only a subset of label sets to support all queries
  - An index built for a label set L can also serve any query set L_q with $L \subseteq L_q$
    - the extra query cost is captured by an elastic factor: the ratio between the numbers of vectors



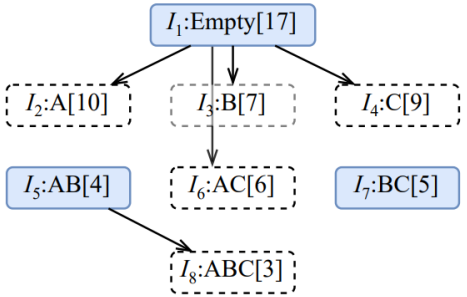
(a): Number of Vectors Associated with Label Set



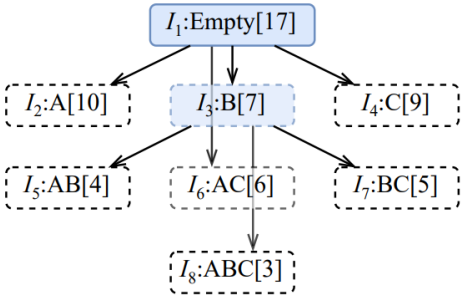
(b): Number of Vectors Matched by Each Query-Label Set

I_1:{Empty, A, B, C, AC}
I_2:{A, AB, AC, ABC}
I_3:{B, AB, BC, ABC}
I_4:{C, AC, BC, ABC}
I_5:{AB, ABC}
I_6:{AC, ABC}
I_7:{BC, ABC}
I_8:{ABC}

(c): The Index Sharing with an Elastic Factor of 0.3



(d): Greedy Result

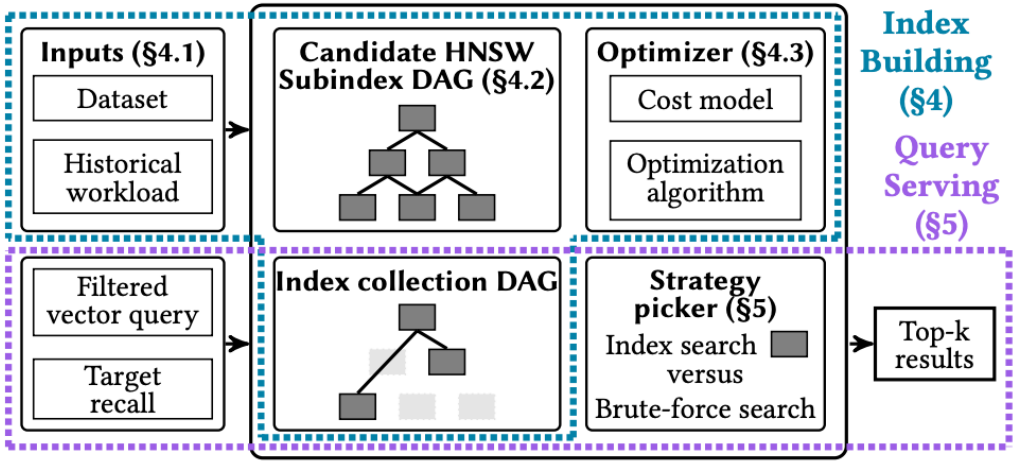


(e): Optimal Result

[1] ELI: Yang et al., Elastic Index Selection for Label-Hybrid AKNN Search. **VLDB 2026**.

Workload-Aware Index Collection

- *SIEVE* builds a collection of indexes based on historical query filters, each tailored to different predicate forms or selectivity ranges
  - To construct this collection under a memory budget
    - It proposes an analytical model capturing the relationships among index size, search time, and recall
    - It uses this model to guide index selection and parameterization jointly

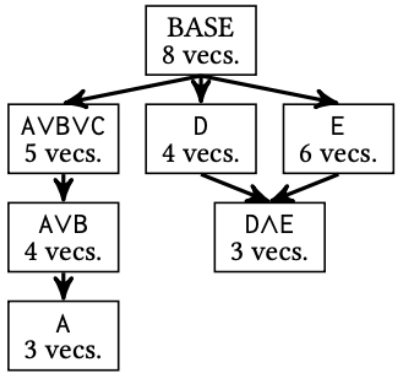


Attributed vec. dataset

Vecs. ($\mathcal{V}$)	attrs ($\mathcal{A}$)
	{A, E}
	{A, E}
	{A, E}
	{B, D}
	{C, F}
	{D, E}
	{D, E}
	{D, E}

Historical Query Filters

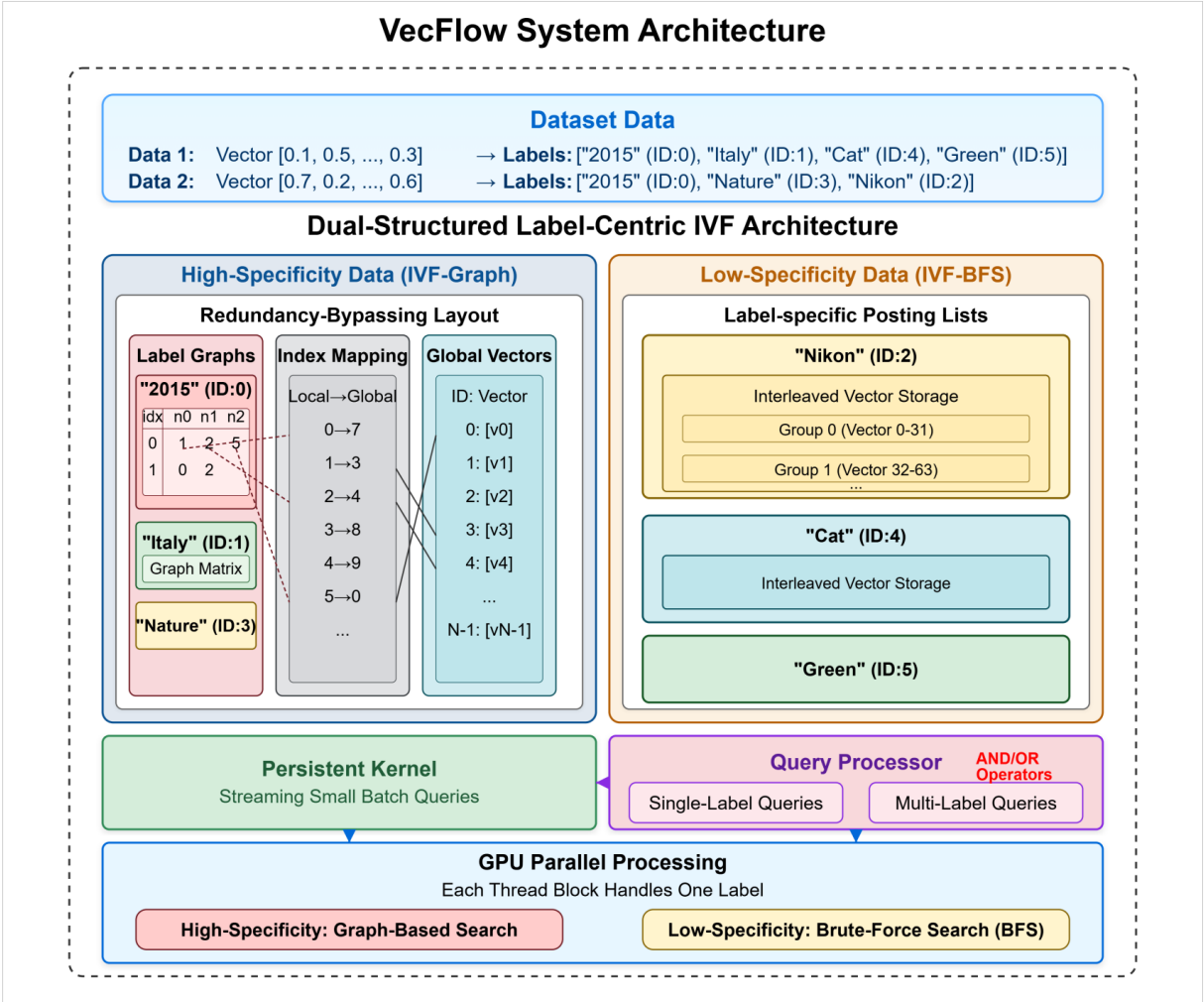
Filter ($\mathcal{F}$)	Count (C)
A	2
A∨B	1
A∨B∨C	3
D	2
E	3
D∧E	1



[1] Li et al., SIEVE: Effective Filtered Vector Search with Collection of Indexes. **VLDB 2025**.

GPU-Oriented Indexing

- VecFlow starts from an IVF-based index, it partitions posting lists according to their label distribution patterns
  - For groups with highly selective labels, VecFlow builds GPU-friendly graph indexes to accelerate candidate retrieval
  - for the remaining groups, it falls back to brute-force scans over the posting lists



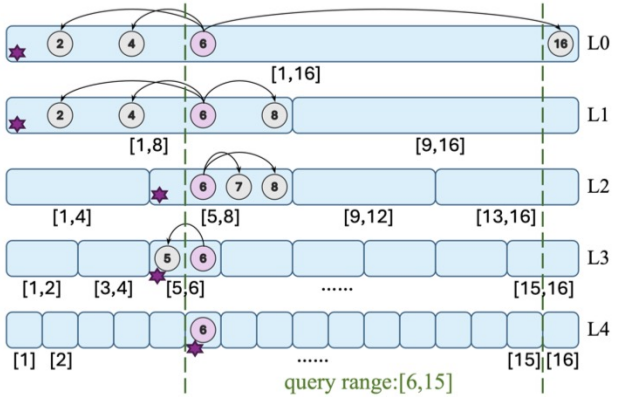
[1] VecFlow: Xi et al., VecFlow: A High-Performance Vector Data Management System for Filtered-Search on GPUs. **SIGMOD 2025**.

Tutorial Overview

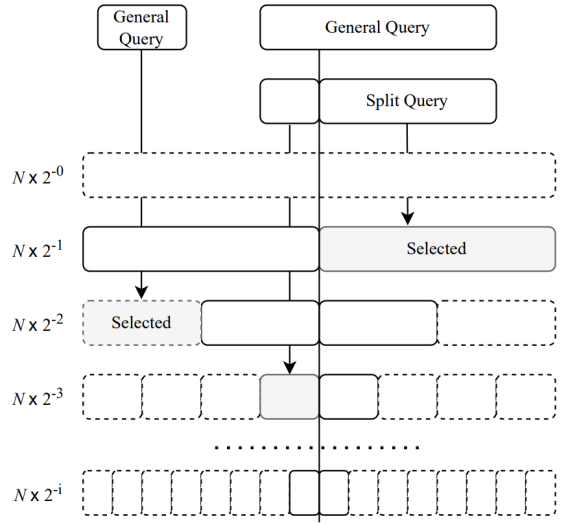
- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
  - **Part 4.1**: Universal Index Approaches
  - **Part 4.2**: Dedicated Index Approaches
    - **Part 4.2.1**: Categorical Attributes
    - **Part 4.2.2**: Numerical Attributes
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

Range-Dedicated Index Decomposition

- SeRF builds a segment graph by sorting objects according to their numerical attributes
  - Build compressed graph indexes for a large collection of ranges
- SuperPostfiltering defines multiple overlapping ranges, building one index for each range
  - Answer a query by choosing a covering range and then applying post-filtering
- iRangeGraph organizes the numerical domain into a segment tree
  - Each tree node corresponds to a range and contains an index
- ESG reduces the number of required subranges to at most two
  - Allow the controlled inclusion of out-of-range points during the search
  - Using post-filtering after search



iRange

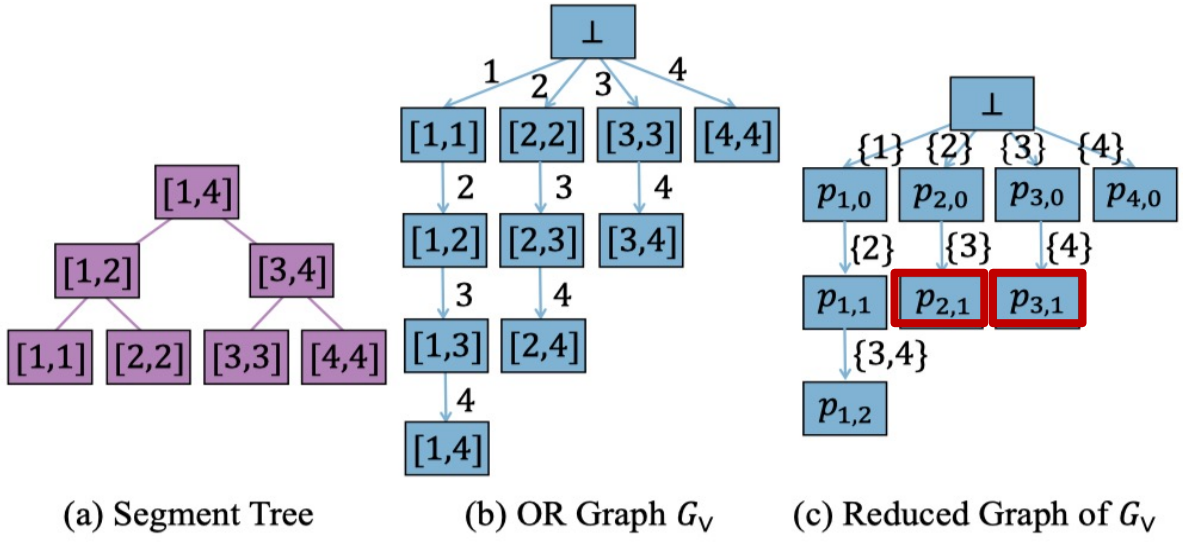


ESG

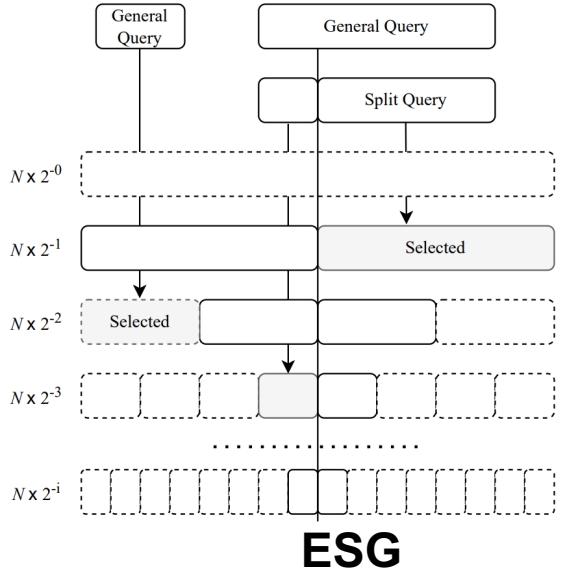
[1] SERF: Zuo et al., SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. **SIGMOD 2024**.
[2] SuperPostfiltering: Engels et al., Approximate nearest neighbor search with window filters. **ICML 2024**.
[3] iRange: Xu et al., iRangeGraph: Improvising Range-dedicated Graphs for Range-filtering Nearest Neighbor Search. **SIGMOD 2025**.
[4] ESG: Yang et al., ESG: Elastic Graphs for Range-Filtering Approximate *k*-Nearest Neighbor Search. **ArXiv 2025**.

Range-Dedicated Index Decomposition

- ESG reduce the number of required subranges to at most two
- 💡**IDEA**: Utilizing OR graph proposed in UniFilter
  - Reduce the number of required subranges to at most two without post-filtering
    - $p_{i,j}$ contains vectors whose attributes are in range $[i, i + 2^j)$.
    - For an arbitrary range $[l, r]$, there are two nodes $p_{l,m}$ and $p_{r-2^m+1,m}$ covering it (e.g., for query range $[2,4]$), where 2^m is the largest power of two that does not exceed $r - l + 1$.



(a) Segment Tree (b) OR Graph G_V (c) Reduced Graph of G_V

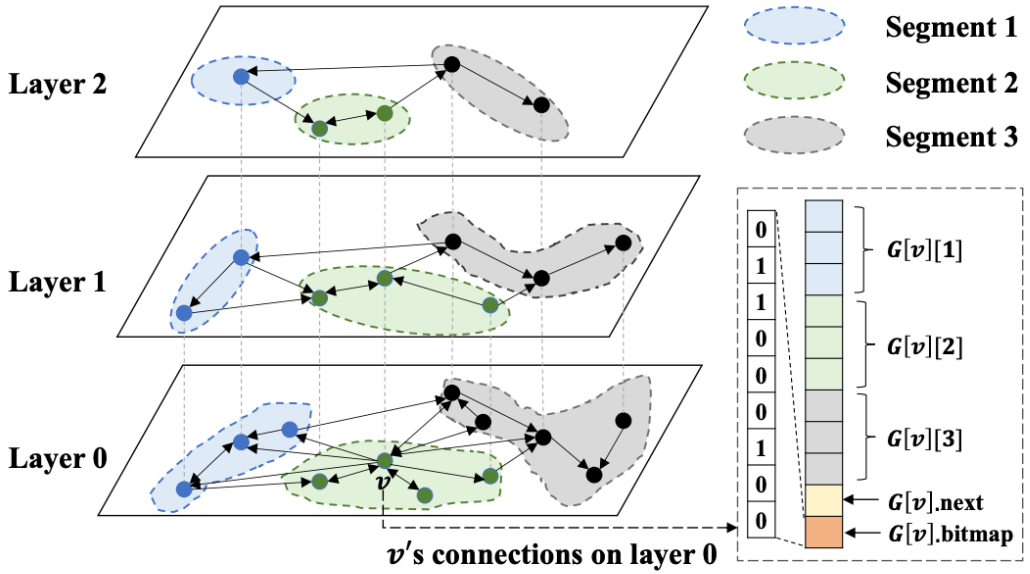


ESG

[1] ESG: Yang et al., ESG: Elastic Graphs for Range-Filtering Approximate *k*-Nearest Neighbor Search. **ArXiv 2025**.
[2] UniFilter: Xie et al., Beyond Vector Search: Querying With and Without Predicates. **SIGMOD 2026**.

Unified Multi-Strategy Frameworks

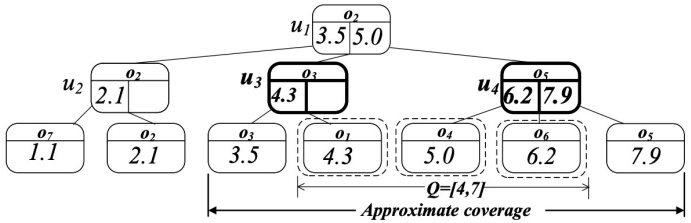
- UNIFY supports pre-filtering, post-filtering, and hybrid filtering within a single PG
  - SIG segments the dataset based on attribute values and theoretically guarantees that the PG of objects from any combination of segments is a connected subgraph of SIG.
  - HSIG can approximately ensure that the HNSW of objects from any combination of segments is a sub-graph of HSIG, while achieving logarithmic hybrid filtering complexity.



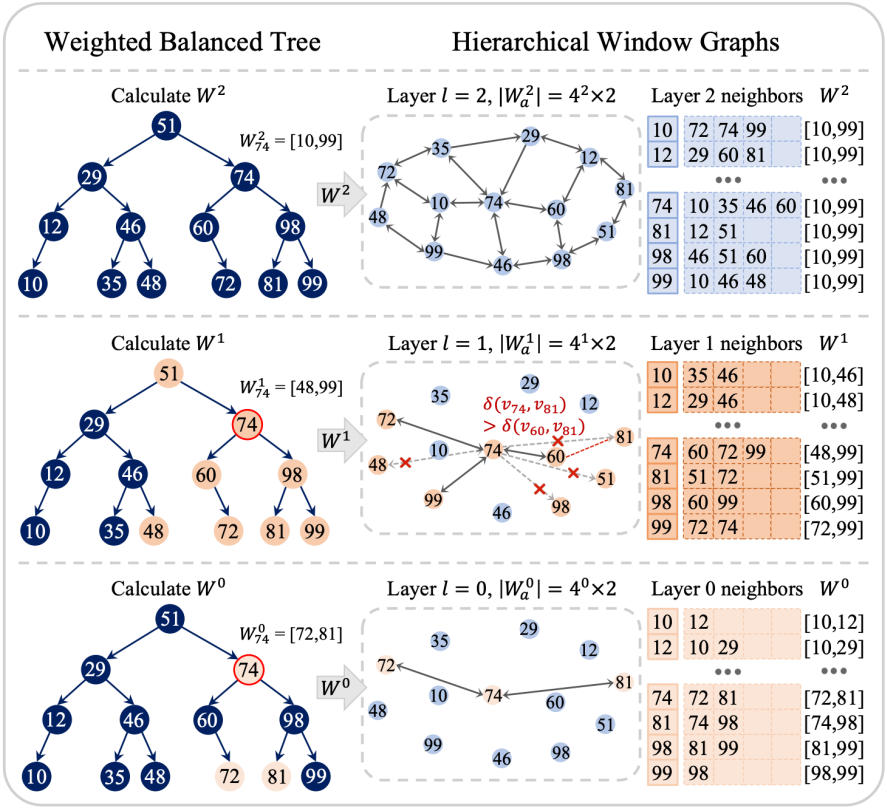
[1] UNIFY: Liang et al., UNIFY: Unified Index for Range Filtered Approximate Nearest Neighbors Search. **VLDB 2025**.

Maintenance-Oriented Methods

- **Range Filter on Streaming Data**
  - DSG supports range filtering with arbitrary vector insertions by encoding the query ranges for which each graph edge is active as 2D rectangles, using a rectangle tree to compactly identify and maintain these range-dependent edges.
- **Support Maintenance**
  - DIGRA Built on a B-tree style organization of numerical attributes, it supports insertions and deletions.
  - RangePQ stores range-aware quantization posting lists in a tree for dynamic maintenance.
  - WoW maintains hierarchical proximity graphs over attribute windows of exponentially increasing sizes, enabling fully incremental construction and selectivity-aware range search.



DIGRA

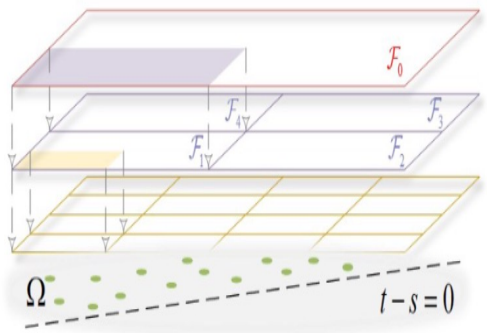


WoW

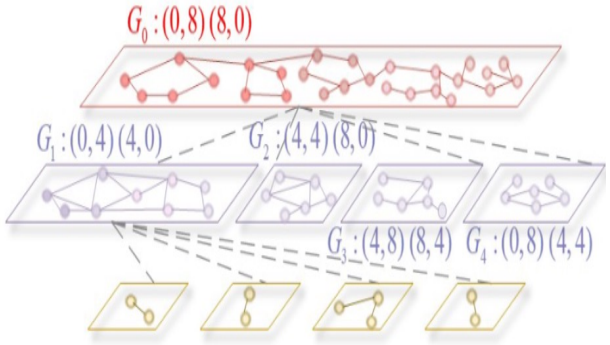
[1] DSG: Peng et al., Dynamic Range-Filtering Approximate Nearest Neighbor Search. **VLDB 2025**.
[2] DIGRA: Jiang et. al., DIGRA: A Dynamic Graph Indexing for Approximate Nearest Neighbor Search with Range Filter. **SIGMOD 2025**.
[3] RangePQ: Zhang et al., Efficient Dynamic Indexing for Range Filtered Approximate Nearest Neighbor Search. **SIGMOD 2025**.
[4] WoW: Wang et al., WoW: A Window-to-Window Incremental Index for Range-Filtering Approximate Nearest Neighbor Search. **SIGMOD 2026**.

ANN Search With Interval Filters

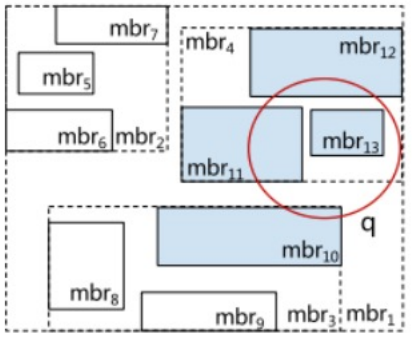
- **Interval Filtering**
  - Hi-PNG transforms each object interval into a point in a two-dimensional space, and transforms the query interval into a query rectangle.
  - It builds a hierarchical interval-partition navigating graph over this transformed space for identifying the relevant interval partitions.
- **Spatial-Range Filtering**
  - Mesh uses R-tree to partition the data.
  - It proposes a workload-aware index to optimize query efficiency while ensuring memory overhead below a predefined constraint via omit some index construction.



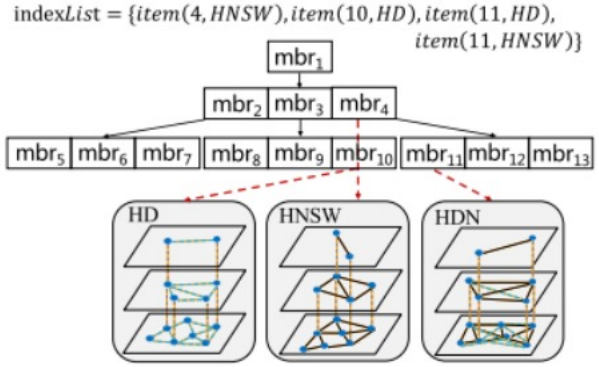
(a) Interval Space Partition



(b) *Hi-PNG*



(a) Data partition

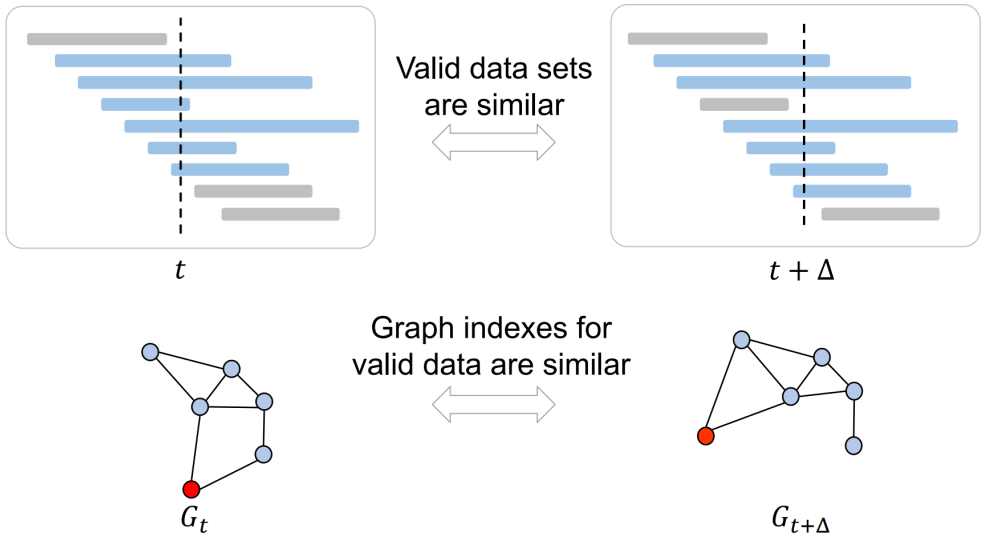


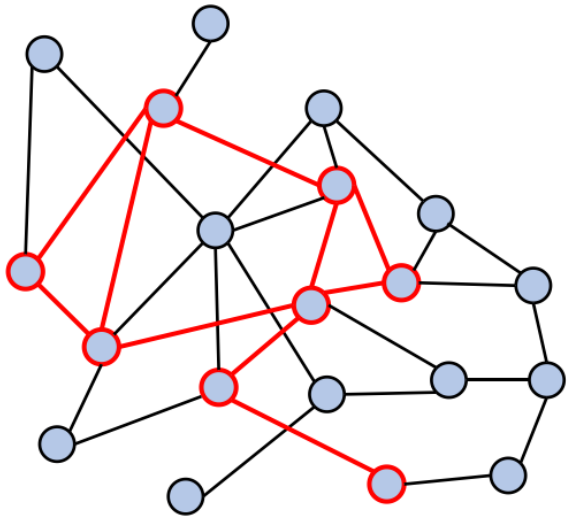
(b) Mesh structure

[1] Hi-PNG: Yang et al., Hi-PNG: Efficient Interval-Filtering ANNS via Hierarchical Interval Partition Navigating Graph. **KDD 2025**.
[2] Mesh: Song et al., Efficient top-k spatial-range-constrained approximate nearest neighbor search on geo-tagged high-dimensional vectors. **VLDBJ 2025**.

ANN Search With Timestamp Filters

- **Timestamp Filtering**
  - TSANN builds a unified graph index for all historical timestamps
    - Aggregate the all per-timestamp graph indexes into a single graph
      - Valid vector sets at nearby timestamps largely overlap
      - A point can appear in neighbor lists for multiple timestamps
    - Search: timestamp-aware greedy routing

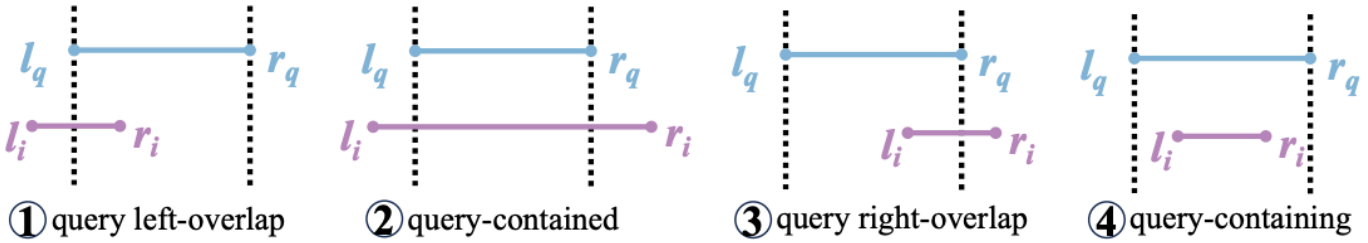




[1] TSANN: Wang et al., Timestamp Approximate Nearest Neighbor Search over HighDimensional Vector Data. **ICDE 2025**.

ANN Search With Arbitrary Range Filters

- **Range-Range Filtering**
  - MSTG supports four atomic range-range predicates or a disjunction of them.
  - It builds a labeled multi-segment tree graph
    - Efficiently handles arbitrary RR predicates by avoiding traversal through non-predicate-satisfied nodes
    - Keeps equivalent index size and construction time to iRangeGraph



Problems	Object Attribute	Query Attribute	RR Predicate	QPS at Recall@10=0.95 on Gist dataset when selectivity is 10%			
				iRangeGraph [32]	Hi-PNG [34]	TS-Graph [27]	MSTG (ours)
RFANN [32]	point-valued ($l_i = r_i$)	range-valued	$[l_i, r_i] \subseteq [l_q, r_q]$ (④)	752.997	127.238	-	753.215
IFANN [34]	range-valued	range-valued	$[l_i, r_i] \subseteq [l_q, r_q]$ (④)	-	156.790	-	873.672
TSANN [27]	range-valued	point-valued ($l_q = r_q$)	$[l_q, r_q] \subseteq [l_i, r_i]$ (②)	-	-	238.851	925.845
RRANN (ours)	range-valued	range-valued	arbitrary	-	-	-	604.572

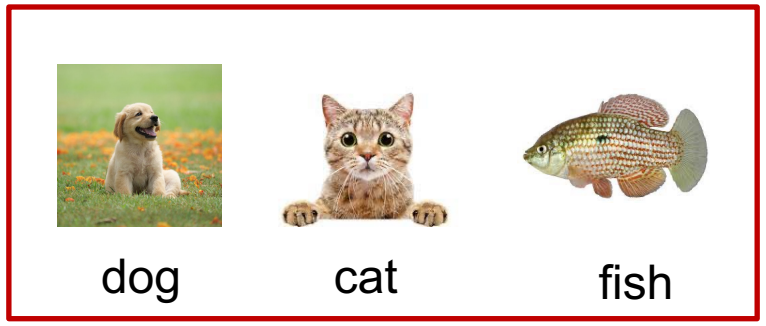
[1] MSTG: Liu et al., Generalized Range Filtering Approximate Nearest Neighbor Search: Containment and Overlap. **KDD 2026**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
  - **Part 5.1**: Exact Similarity Join
  - **Part 5.2**: Approximate Similarity Join
- **Part 6**: Open Challenges

Similarity Join

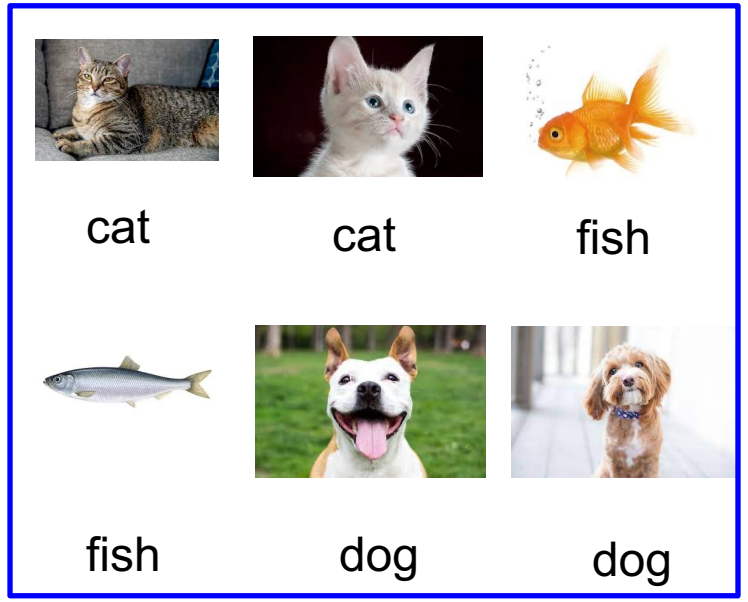
- Similarity join on such objects/texts is to join the similar vectors in the vector database.
  - Auto-tagging, Tags the animal images based on my sample tags.



Animal tags

Untagged Animal Images

```
SELECT Animals.image, Animal_Tags.name
FROM Animals JOIN Animal_Tags
ON DISTANCE(Animals.image_embedding,
Animal_Tags.image_embedding) < ϵ;
```



Animal tags

Vector Similarity Join

- **ϵ-similarity Join**
  - Given two sets of points, $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_m\}$, in Euclidean d-dimensional space, denoted as R^d, for $d > 0$, the ϵ-similarity join $X \bowtie Y$ is defined as $X \bowtie Y = \{(x_k, y_l) \in X \times Y \mid \delta(x_k, y_l) \leq \epsilon\}$, where $\delta(x_k, y_l)$ is L_2 norm (i.e., Euclidean distance) between $x_k \in X$ and $y_l \in Y$, and ϵ is a user-given threshold where $\epsilon > 0$.
  - We refer the ϵ-similarity join, as ϵ-similarity self-join, if the two given sets are identical.

- **k-similarity Join**
  - Given two sets of points, $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_m\}$, in Euclidean d-dimensional space, denoted as R^d, for $d > 0$, the k-similarity join $X \bowtie_k Y$ is defined as $X \bowtie_k Y = \{(x_k, y_l) \in X \times Y \mid y_l \in TopK(x_k)\}$, where $TopK(x_k)$ is the set of top-k nearest data points of x_k in Y.
  - We refer the k-similarity join, as k-similarity self-join, if the two given sets are identical.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
  - **Part 5.1**: Exact Similarity Join
  - **Part 5.2**: Approximate Similarity Join
- **Part 6**: Open Challenges

Exact Vector Similarity Join

- **Two steps** for the state-of-the-art methods for exact high-dimensional similarity join
  - Filtering
    - Candidate pairs of vectors that might be part of the final join results are selected through indexing or sorting.
    - E.g., Epsilon Grid Order.
  - Refinement
    - The exact distance between candidate pairs of two vectors is computed.
- Refine the pairs in an order defined by a space-filling curve which dramatically **improves data locality**.

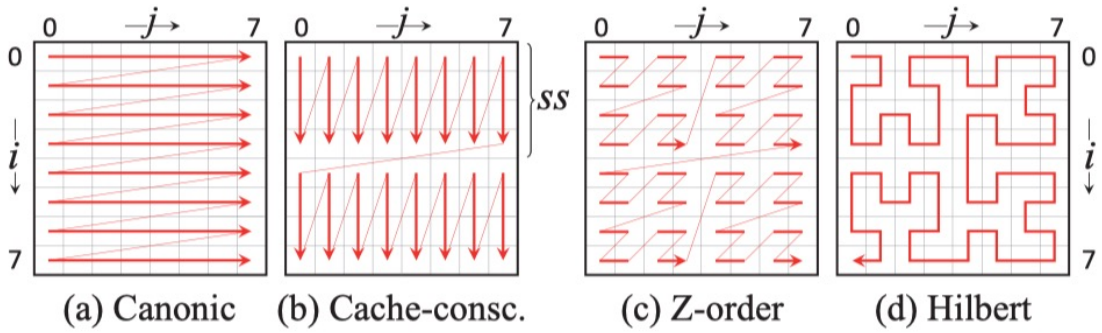
Definition 2.1 (EGO).

$\mathbf{x}_i \leq_{\text{EGO}} \mathbf{x}_j \quad :\Leftrightarrow$ one of the following conditions holds:

(1) $\left\lfloor \frac{x_{i,k}}{\epsilon} \right\rfloor = \left\lfloor \frac{x_{j,k}}{\epsilon} \right\rfloor$ for all k with $0 \leq k < d$, **or**

(2) there exists a dimension D with $0 \leq D < d$ such that

$$\left\lfloor \frac{x_{i,k}}{\epsilon} \right\rfloor = \left\lfloor \frac{x_{j,k}}{\epsilon} \right\rfloor \quad \text{for all } k < D \quad \textbf{and}$$
$$\left\lfloor \frac{x_{i,D}}{\epsilon} \right\rfloor < \left\lfloor \frac{x_{j,D}}{\epsilon} \right\rfloor.$$



[1] Böhm et al., Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-Dimensional Data. **SIGMOD 2001**.
[2] Perdacher et al., Cache-oblivious High-performance Similarity Join. **SIGMOD 2019.**

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
  - **Part 5.1**: Exact Similarity Join
  - **Part 5.2**: Approximate Similarity Join
- **Part 6**: Open Challenges

Approximate Vector Similarity Join

- **Three steps** of **Locality-sensitive Hashing** (LSH) based approaches
  - Projecting each vector into a hash value
  - Executing an equi-join on the pairs that collide under the same hash value to establish candidate join results
  - Exact distance computations are conducted to derive the final join results.

- The **effectiveness** of these approaches heavily depends on the **hash functions** in the first step.

[1] Aumüller et al., Implementing Distributed Similarity Joins using Locality Sensitive Hashing. **EDBT 2022**.
[2] Hu et al., Output-Optimal Massively Parallel Algorithms for Similarity Joins. **TODS 2019**.
[3] Li et al., C2Net: A Network-Efficient Approach to Collision Counting LSH Similarity Join. **TKDE 2019**.
[4] Yu et al., A Generic Method for Accelerating LSH-Based Similarity Join Processing. **TKDE 2017**.

Approximate Vector Similarity Join

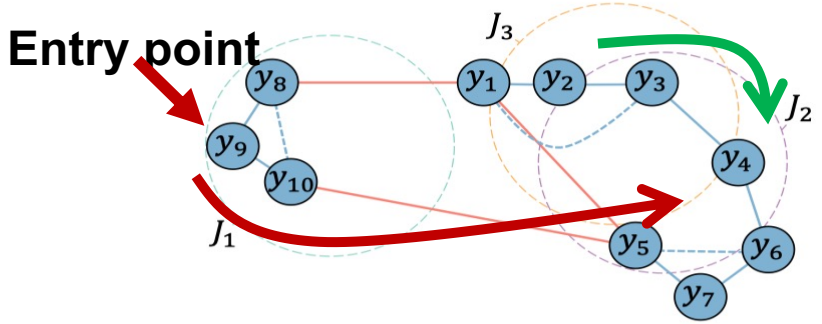
- **Join Window**
  - Let J_k be the join window for $x_k \in X$, such that $J_k = \{y_l \in Y \mid \delta(x_k, y_l) \leq \epsilon\}$.
  - $X \bowtie Y = \{(x_k, y_l) \in X \times Y \mid \delta(x_k, y_l) \leq \epsilon\} = \cup_{x_k \in X} (\{x_k\} \times J_k)$.
- Such approaches use an approximate ϵ-range query to find each join window J_k in Y.
  - Optimize the join processing time by refining the **selection operations:**
    - skipping tuples with zero or few join results by learning-based method (XJoin);
    - bypassing the limitations of a top k-only interface to support ϵ-range search (VBASE).

- They are selection-based approaches.
  - They do not fully leverage the inherent property of the ϵ-similarity join, and utilize the join results from processed data points to accelerate the processing time of other data points that have not been processed.

[1] XJOIN: Yin et al., Xling: A Learned Filter Framework for Accelerating High-Dimensional Approximate Similarity Join. **Arxiv 2024**.
[2] VBASE: Zhang et al., VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. **OSDI 2023**.

SimJoin: Leverage Processed Data

- Leverage the inherent property of the ϵ-similarity join
  - Utilize join results from processed data points to accelerate the processing time of other data points that have not been processed.
- Intuitively, utilizing the join results from a close data point will reduce the nodes traversed by a single ϵ-range query. SimJoin refers this as Join Window Sliding
  - Sliding the processed join window J_i to the join window J_j to be processed next for x_j.
- **Join Window Order Selection**
  - Let $\aleph$ be the join window order over X such that $\aleph = \langle(\kappa_1, x_1), (\kappa_2, x_2), \cdots, (\kappa_n, x_n)\rangle$
  - A pair of (κ_i, x_i) indicates that the join window, J_i, to be processed next for x_i is slide from the join window represented by $\kappa_i \in X$.



[1] SimJoin: Xie et al., Fast Approximate Similarity Join in Vector Databases. **SIGMOD 2025**.

Offline-Online Co-Design

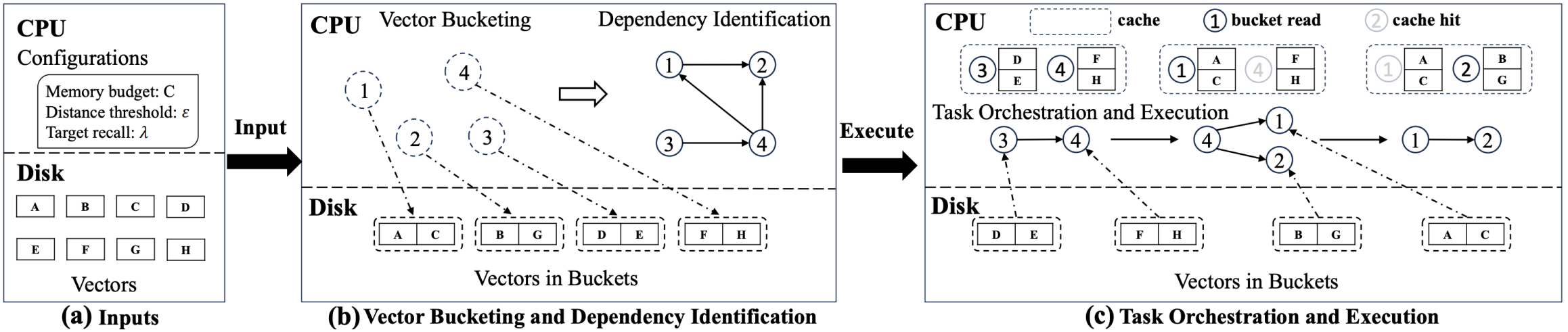
- **Soft Work Sharing**
  - Reuses traversal effort, rather than only reusing final join results.
  - SimJoin only reuses the join results while wasting the traversal results during search
- **Work Offloading**
  - Build a merged query–data graph to encode cross-set proximity offline, avoiding repeated online graph navigation.
- **OOD-Aware Search**
  - Use adaptive BFS–BestFS traversal to bridge disconnected in-range regions for OOD queries.

(a) Separate indexes. (b) Merged index.

[1] Kim et al., Fast Approximate Vector Joins via Offline-Online Co-Design. **Arxiv 2026**.
[2] SimJoin: Xie et al., Fast Approximate Similarity Join in Vector Databases. **SIGMOD 2025**.

Approximate Similarity Join with SSD

- **DiskJoin** targets SSD-resident vector similarity join on a single machine
  - Cluster nearby vectors into disk-resident buckets and build a bucket dependency graph.
  - Prune distant candidate buckets under a target-recall error budget before verification.
  - Reorder bucket-pair tasks to maximize reuse, then use future-aware Belady cache eviction.



(a) Inputs (b) Vector Bucketing and Dependency Identification (c) Task Orchestration and Execution

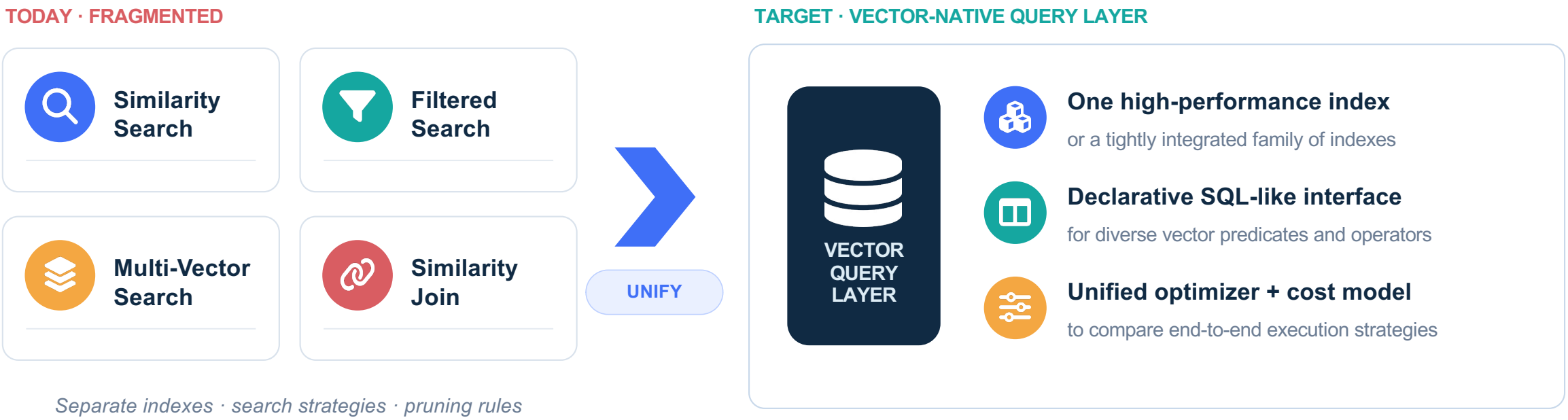
[1] Chen et al., DiskJoin: Large-scale Vector Similarity Join with SSD. **SIGMOD 2026**.

Tutorial Overview

- **Part 1**: Background and Motivation
- **Part 2**: Similarity Search
- **Part 3**: Multi-Similarity Search
- **Part 4**: Filtered Similarity Search
- **Part 5**: Similarity Join
- **Part 6**: Open Challenges

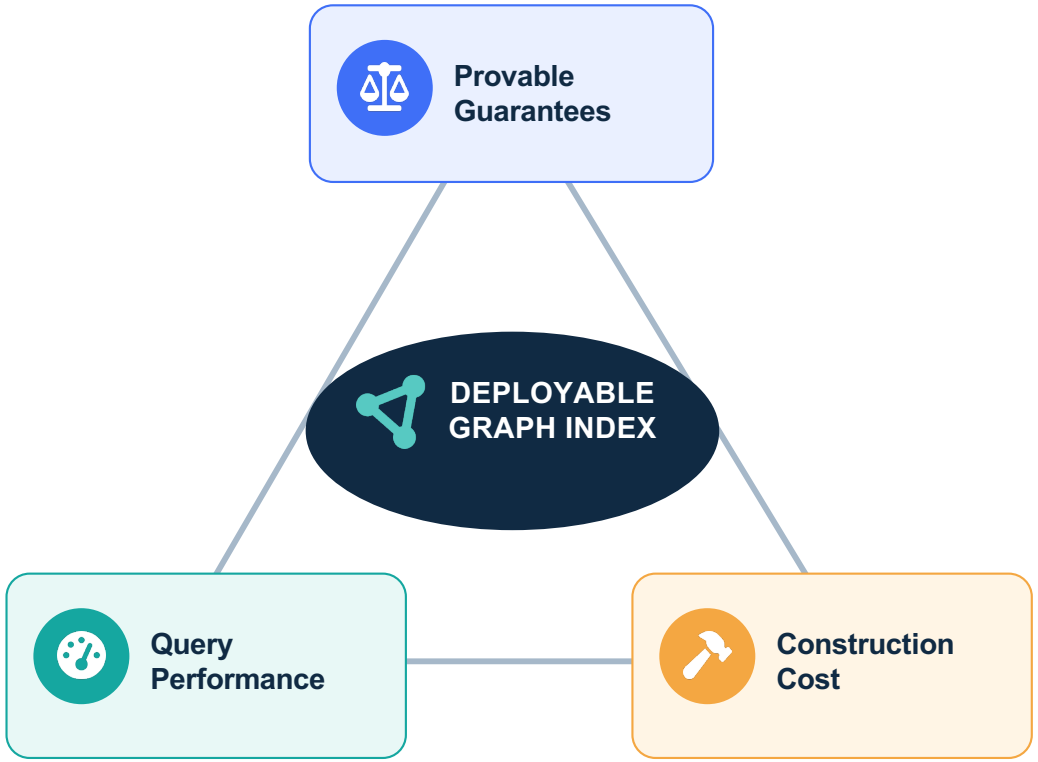
Toward Unified Query Processing

- Today's vector workloads cross boundaries that current engines still treat separately.
  - How should vector predicates be represented?
  - How should approximate and relational operators be co-optimized?
  - What statistics predict end-to-end cost and quality?



Theoretical Guarantees v.s. Practical Performance

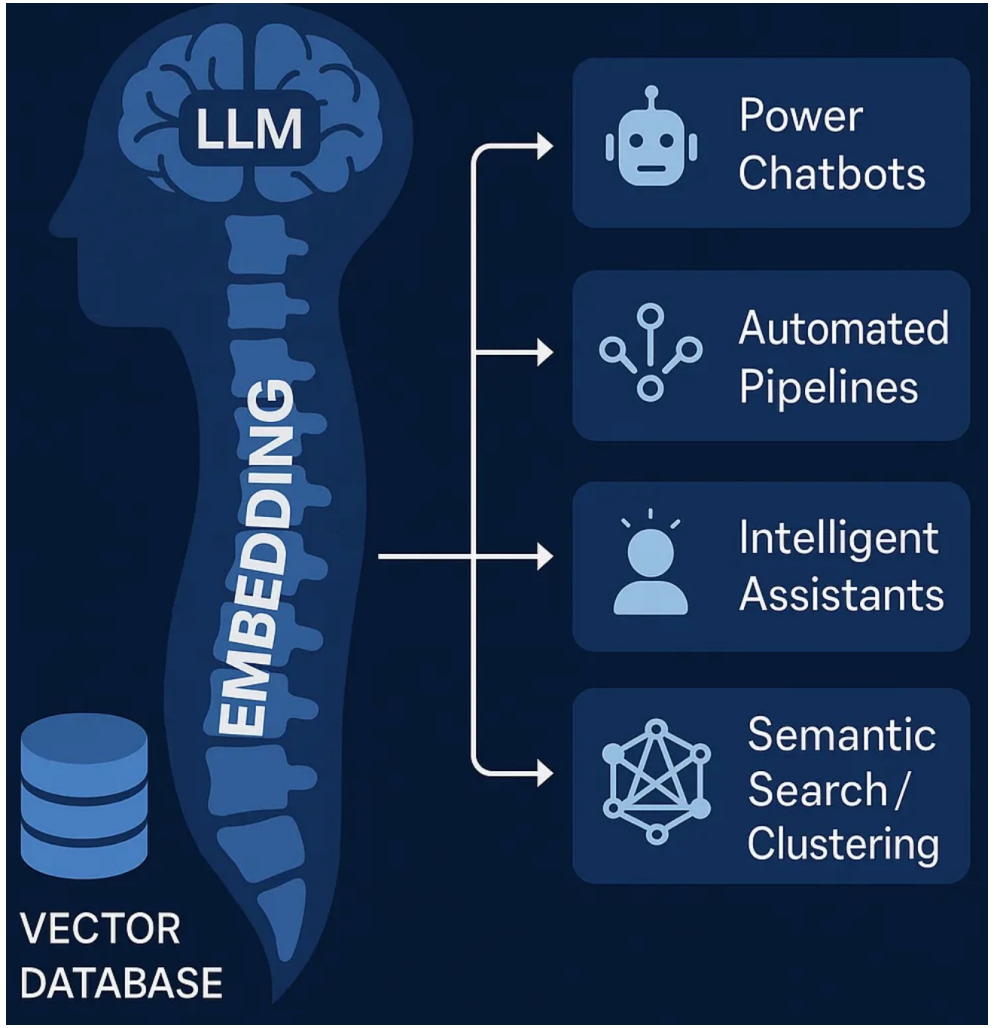
- State-of-the-art graph indexes are highly effective but mostly heuristic.
- Provable designs often require strong assumptions or expensive index cost.
- **Evaluations**
  - Through empirical trade-offs on selected datasets and benchmark suites, using metrics such as recall, QPS, latency, and memory
  - Develop theory-driven or structure-aware evaluation methods
    - Graph properties
    - Navigability measures
    - Robustness indicators
    - Approximation bounds



Optimize Agentic Retrieval Workloads

- **Vector databases are evolving from ANN engines to a query execution layer for LLM pipelines**
  - Declarative document processing
  - Semantic operators
  - Cost-quality optimization

- **Vector databases are increasingly used in workloads in agentic systems**
  - Multi-query workflows
  - Memory + context management
  - Orchestration across tools/data/models
  - Reliability, privacy, and security for agentic retrieval



Some Remarks

- There are still many research problems for us to explore.

More Details

A Survey on Query Processing in Vector Databases

JIADONG XIE, The Chinese University of Hong Kong, China
YINGFAN LIU, Xidian University, China
JEFFREY XU YU, The Hong Kong University of Science and Technology (Guangzhou), China

High-dimensional vectors have become a fundamental data representation in modern applications, such as information retrieval and large language model systems, making vector databases and their query processing an essential research area. While approximate nearest neighbor search has long been the central primitive, modern vector workloads increasingly involve richer query types, including filtered similarity search, multi-vector similarity search, and similarity join. These developments substantially expand the design space of vector query processing and make it harder to obtain a clear and structured view of existing techniques. This survey presents a comprehensive review of query processing in vector databases. We first formalize four query types: similarity search, filtered similarity search, multi-vector similarity search, and similarity join. We then organize existing studies under a unified taxonomy. In particular, we review proximity graphs and quantizations, the two state-of-the-art approaches for similarity search, together with related directions such as distance computation, hard-query processing, and secure search. We further summarize universal and dedicated approaches for filtered similarity search, different methods for multi-vector similarity search, and both exact and approximate algorithms for similarity join. Through this survey, we provide a structured view of current approaches, highlight their connections and differences, and discuss open challenges and future directions.

Thank You!

Our Survey Paper
https://engrxiv.org/preprint/view/7009

Website of Tutorial Materials
https://vdb-query-processing.github.io/