ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data

11/12/2026

Tzu-I Ho,
CS 848 – Fall 2026



UNIVERSITY OF WATERLOO

Hybrid Search

Hybrid Search combines:

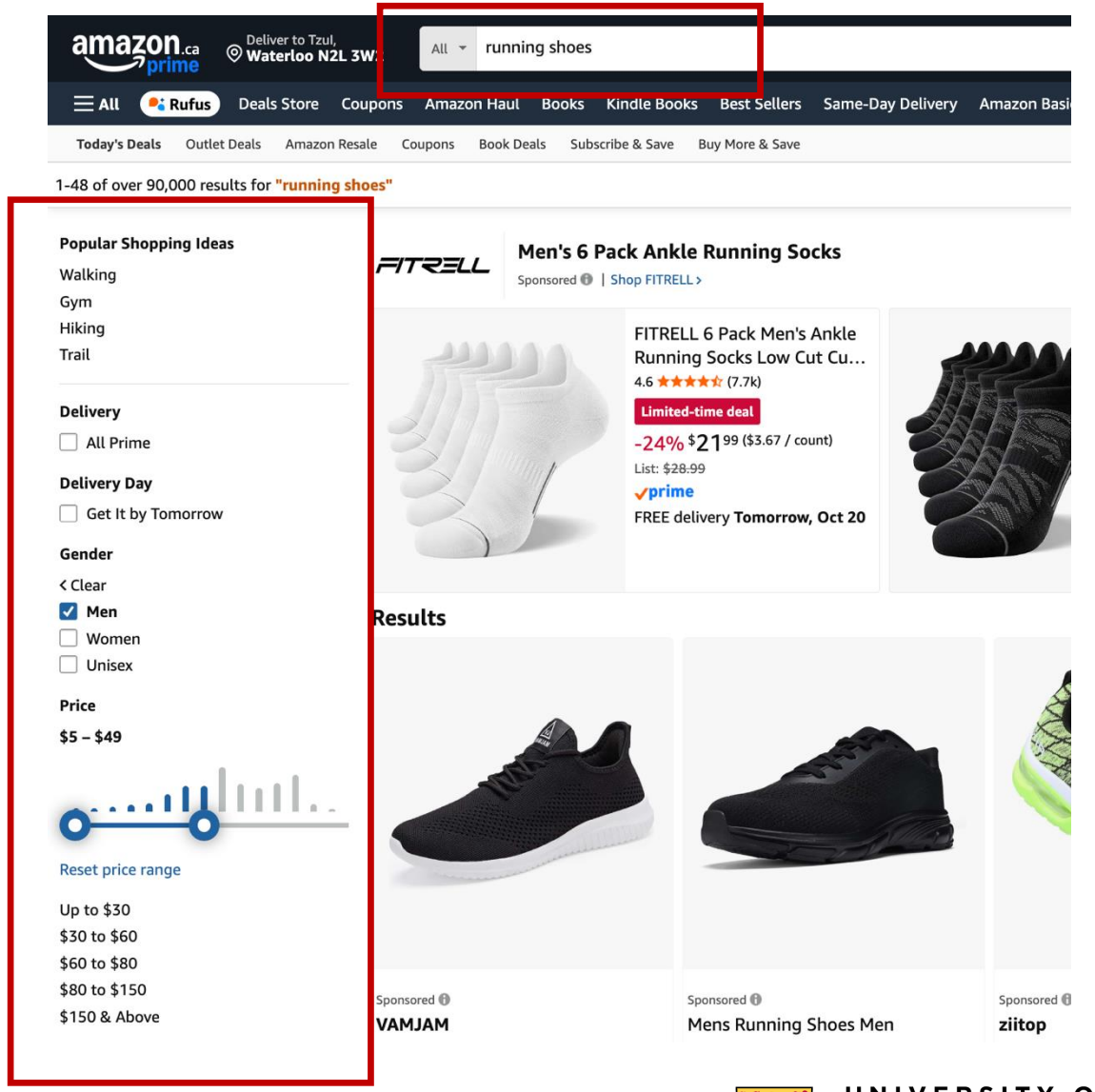
- Vector similarity search
  (e.g., text or image embeddings)
- Structured predicates
  (e.g., gender = Men, price < $50)

↓

Find the Top-K nearest neighbors among items satisfying the predicates

TEXT QUERY → QUERY VECTOR

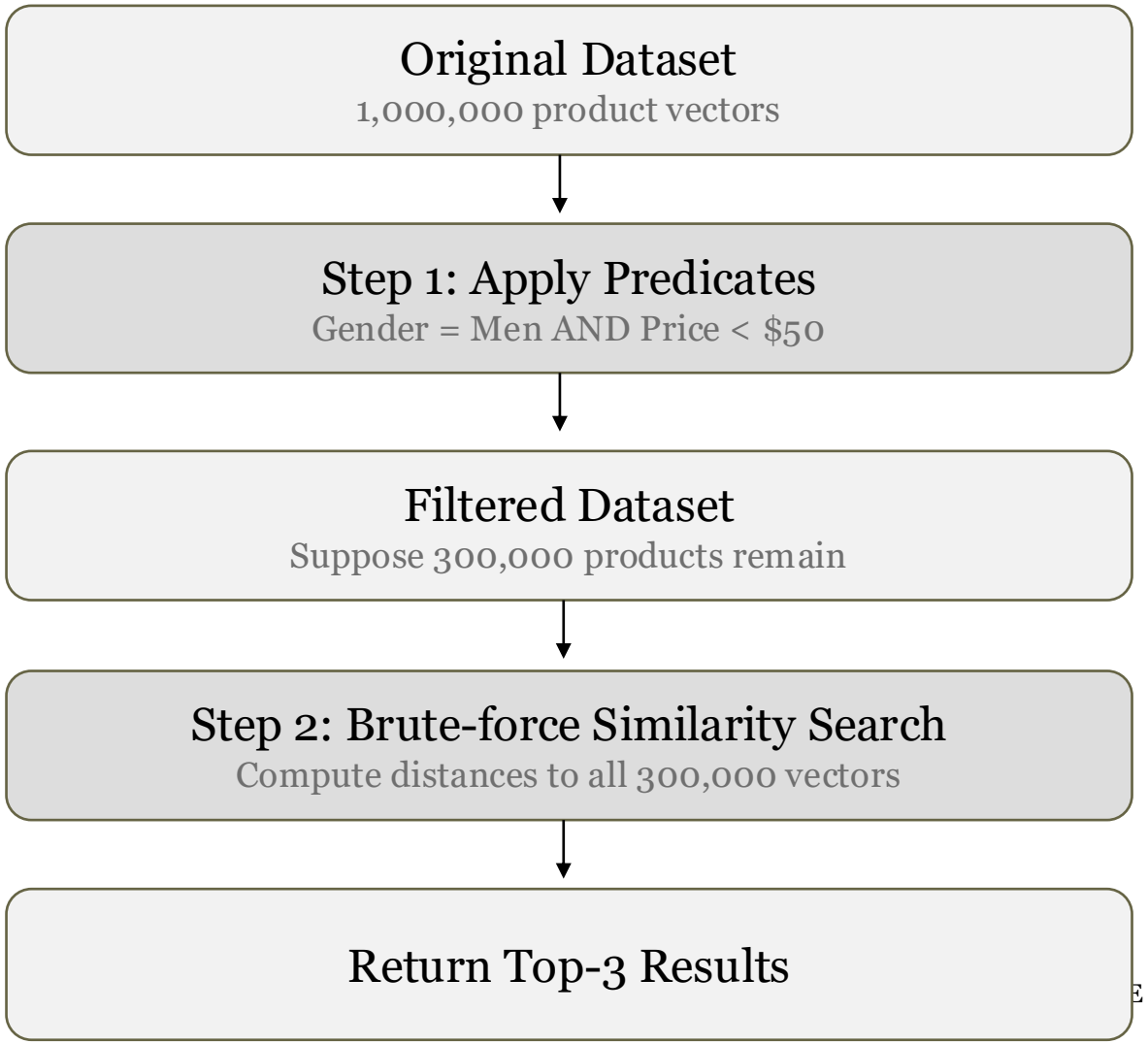
STRUCTURED PREDICATES



UNIVERSITY OF WATERLOO

Pre-filtering: Filter First, Search Later

Example: Find the Top-3 running shoes for Men under $50



$n = 1{,}000{,}000$
$s = 0.3$
$K = 3$

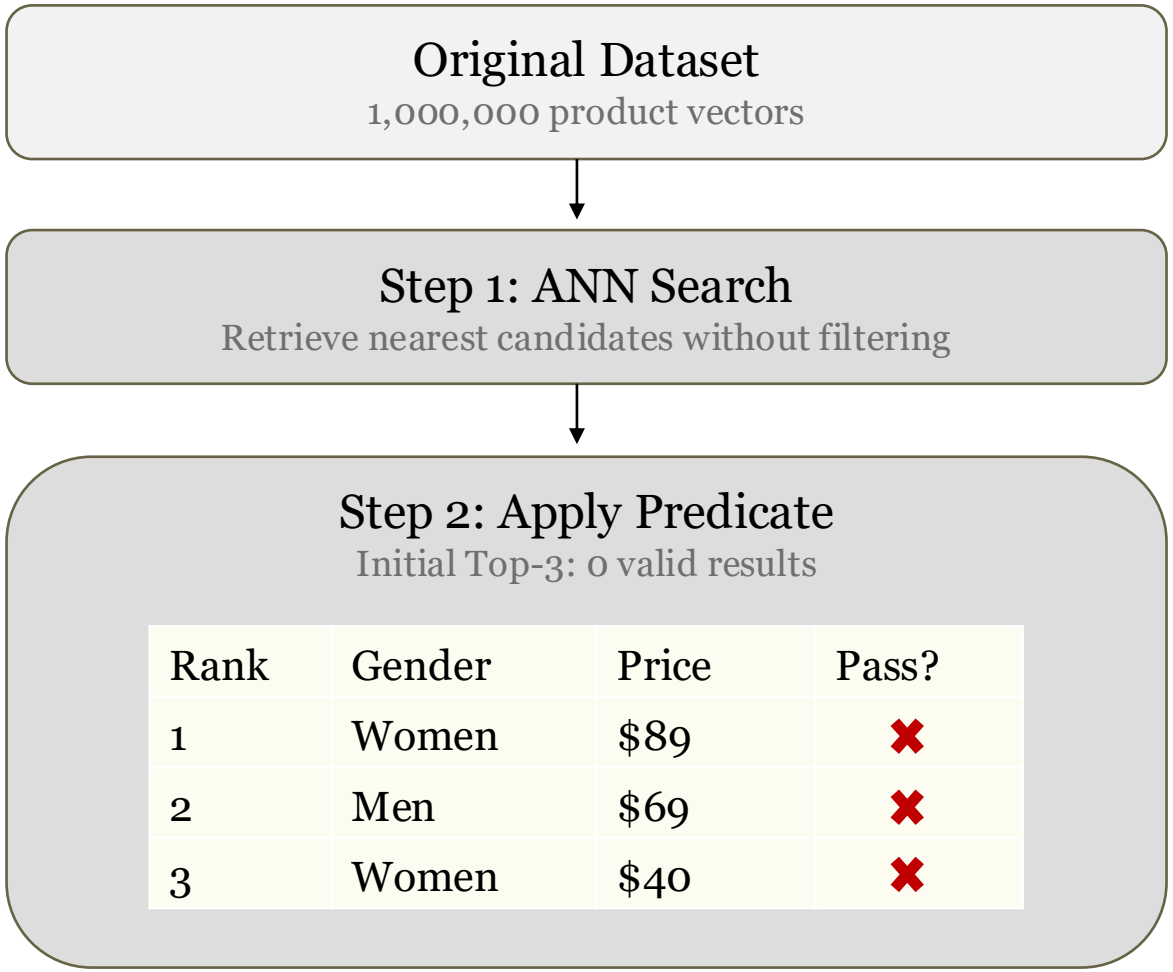
Bottleneck: Too many distance computations
Search cost grows with the number of matching vectors.

Time Complexity: $O(sn + k)$

UNIVERSITY OF WATERLOO

Post-filtering: Search First, Filter Later

Example: Find the Top-3 running shoes for Men under $50



Rank	Gender	Price	Pass?
1	Women	$89	✖
2	Men	$69	✖
3	Women	$40	✖

UNIVERSITY OF WATERLOO

Post-filtering: Search First, Filter Later

Example: Find the Top-3 running shoes for Men under $50

Original Dataset
1,000,000 product vectors

↓

Step 1: ANN Search
Retrieve nearest candidates without filtering

↓

Step 2: Apply Predicate
Initial Top-3: 0 valid results

Rank	Gender	Price	Pass?
1	Women	$89	✖
2	Men	$69	✖
3	Women	$40	✖

↓

Expand ANN search scope

Rank	Gender	Price	Pass?
1	Women	$89	✖
2	Men	$69	✖
3	Women	$40	✖
4	Men	$39	○
5	Women	$29	✖
6	Men	$35	○
7	Men	$45	○

Bottleneck: Expensive search expansion
Many nearest candidates may fail the predicates.

UNIVERSITY OF WATERLOO

Limitations of Existing Methods

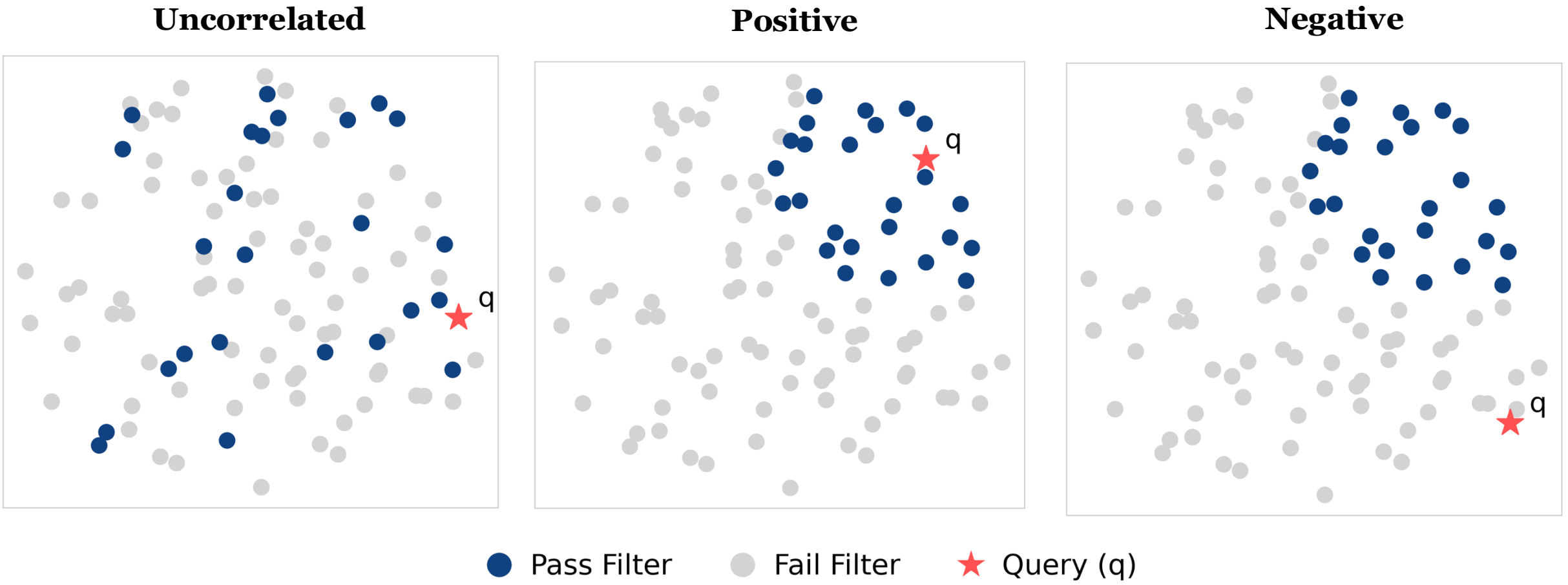
	Pre-filtering	Post-filtering
Works well when	Few vectors pass the filter	Nearby candidates pass the filter
Struggles when	Many vectors pass the filter	Few nearby candidates pass the filter

Neither approach is consistently efficient across all query conditions.
Performance depends on the number and distribution of matching vectors.

UNIVERSITY OF WATERLOO

Query Correlation: Why Selectivity Is Not Enough

Same filter selectivity, different spatial distributions relative to the query

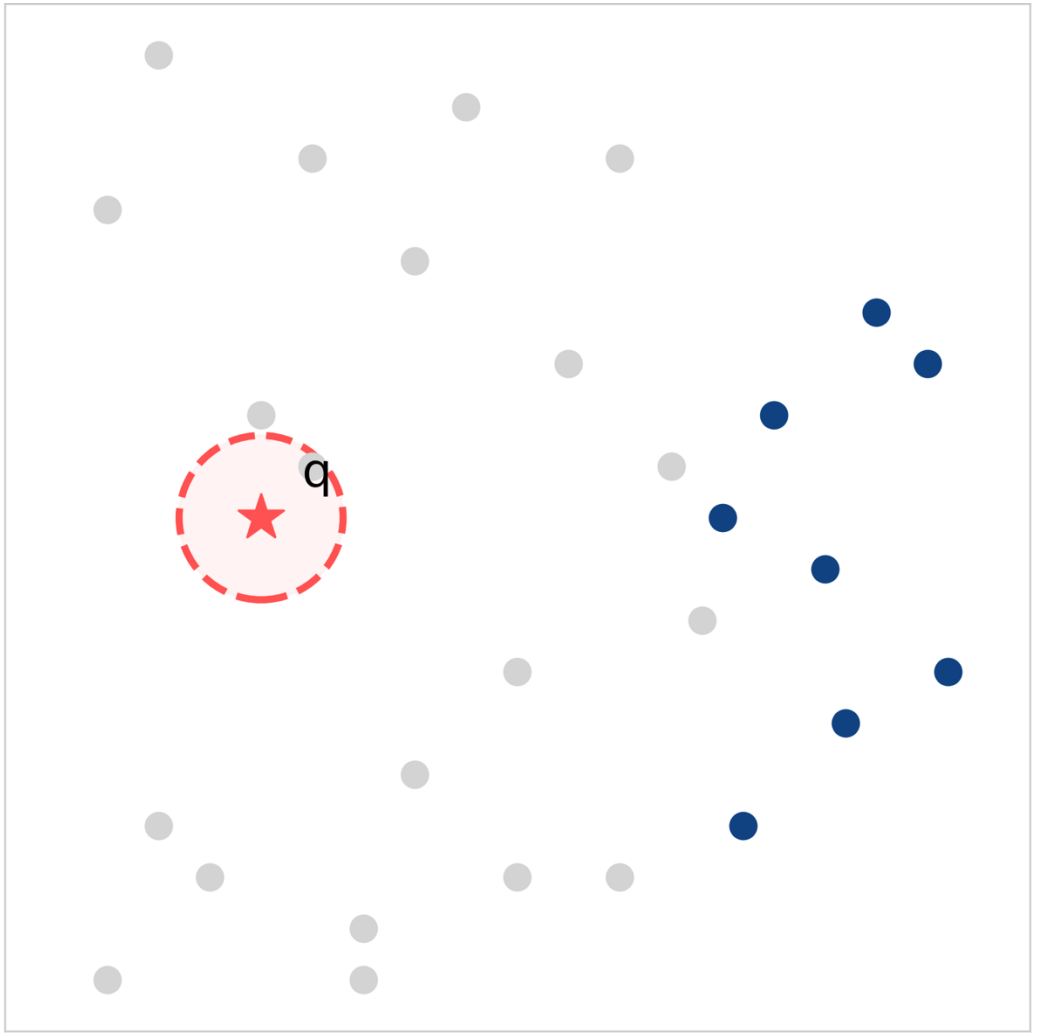


Same selectivity does not imply the same Post-filtering cost

UNIVERSITY OF WATERLOO

Impact of Query Correlation on Post-filtering

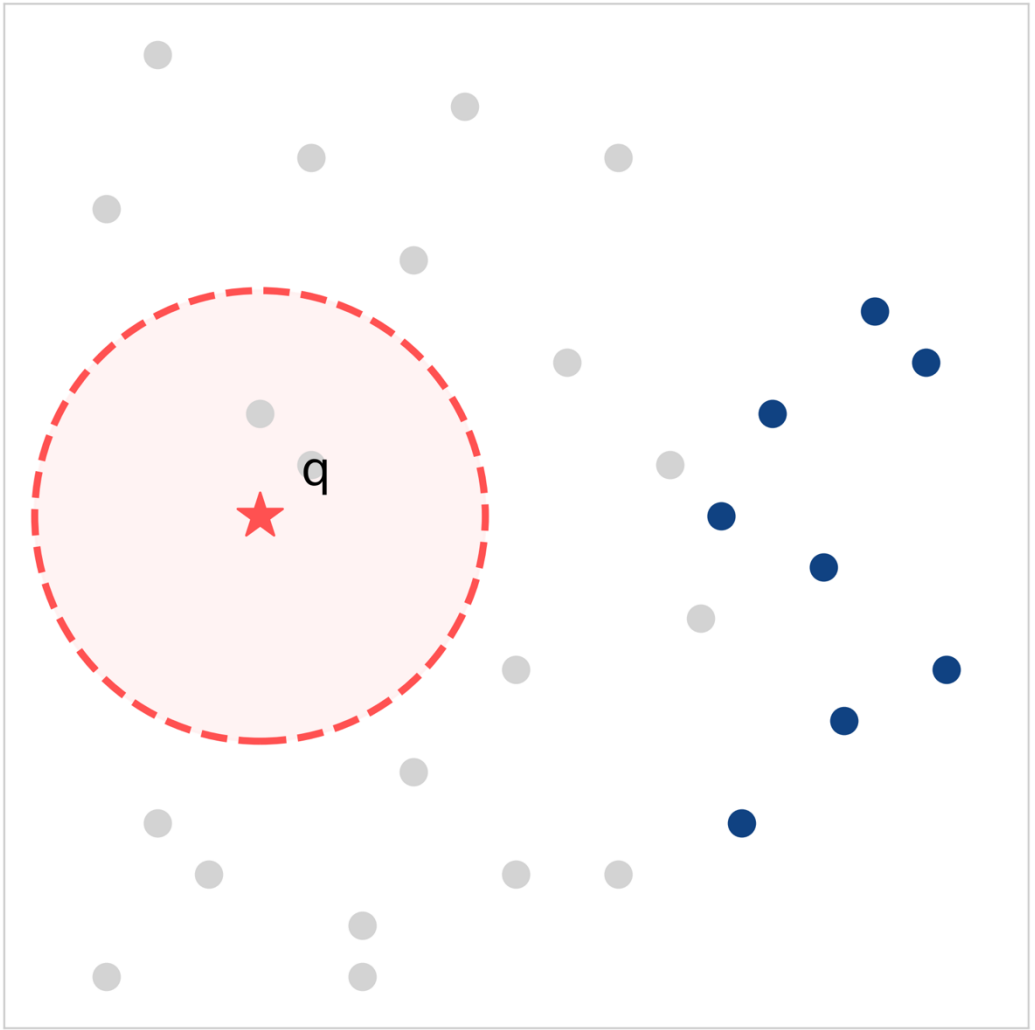
Initial Search



UNIVERSITY OF WATERLOO

Impact of Query Correlation on Post-filtering

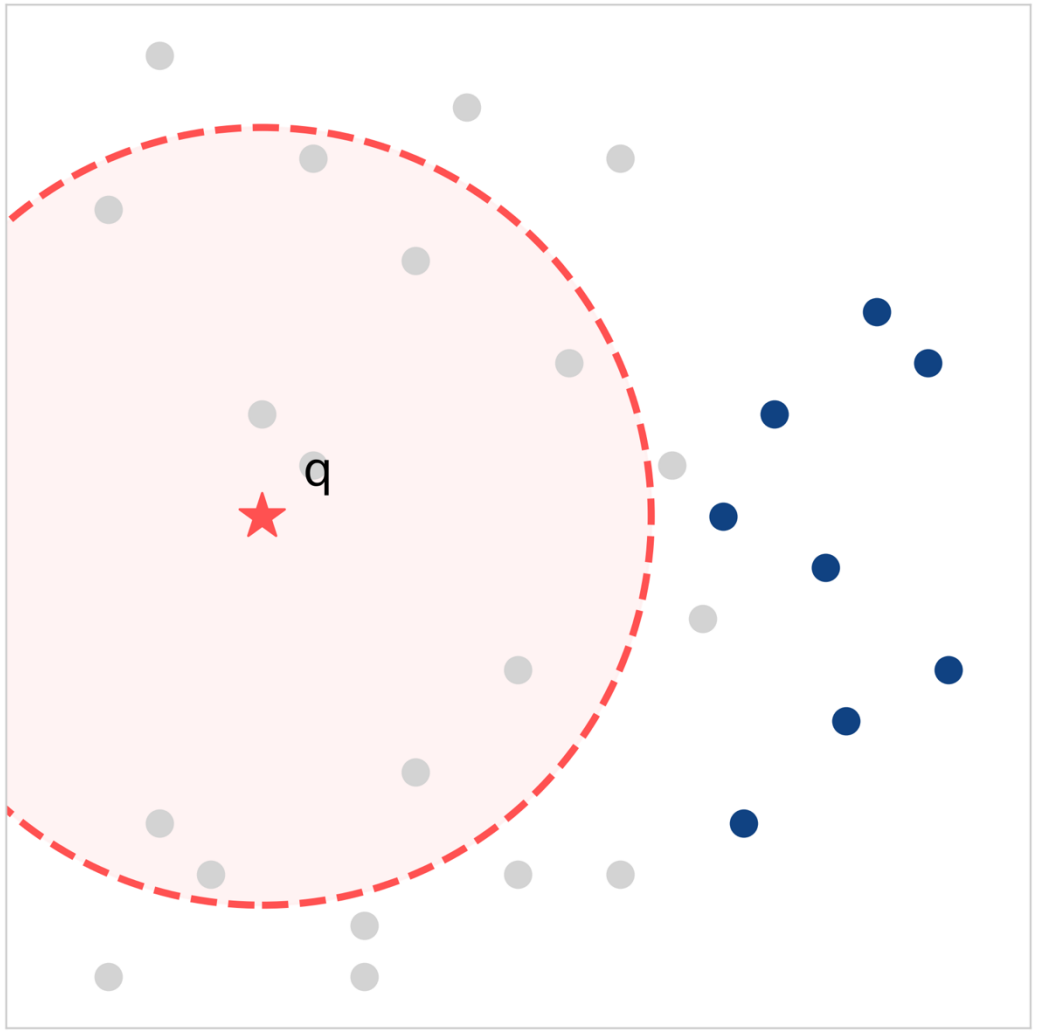
Expand the Search



UNIVERSITY OF WATERLOO

Impact of Query Correlation on Post-filtering

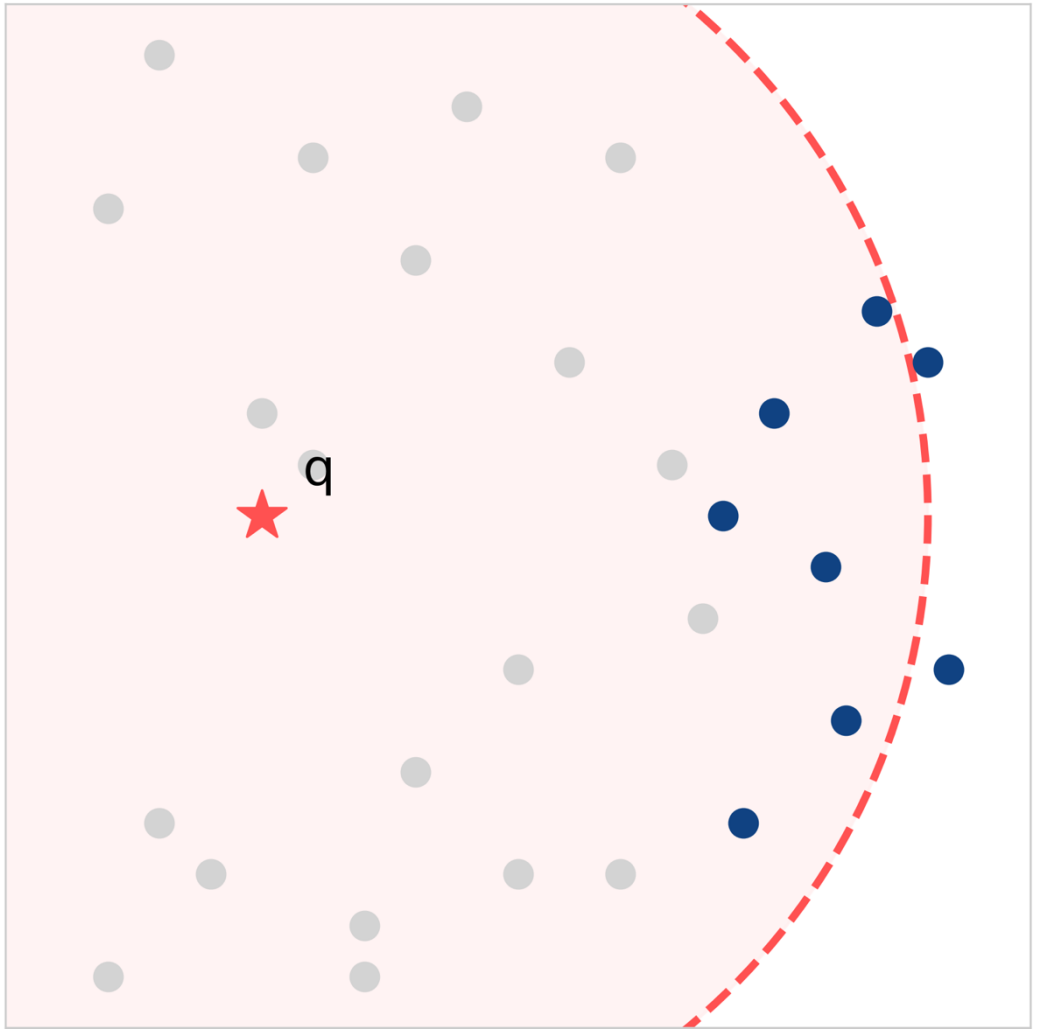
Continue Expanding



UNIVERSITY OF WATERLOO

Impact of Query Correlation on Post-filtering

Negative Query Correlation
→ Large Search Expansion
→ High Search Cost



UNIVERSITY OF WATERLOO

How HNSW Works

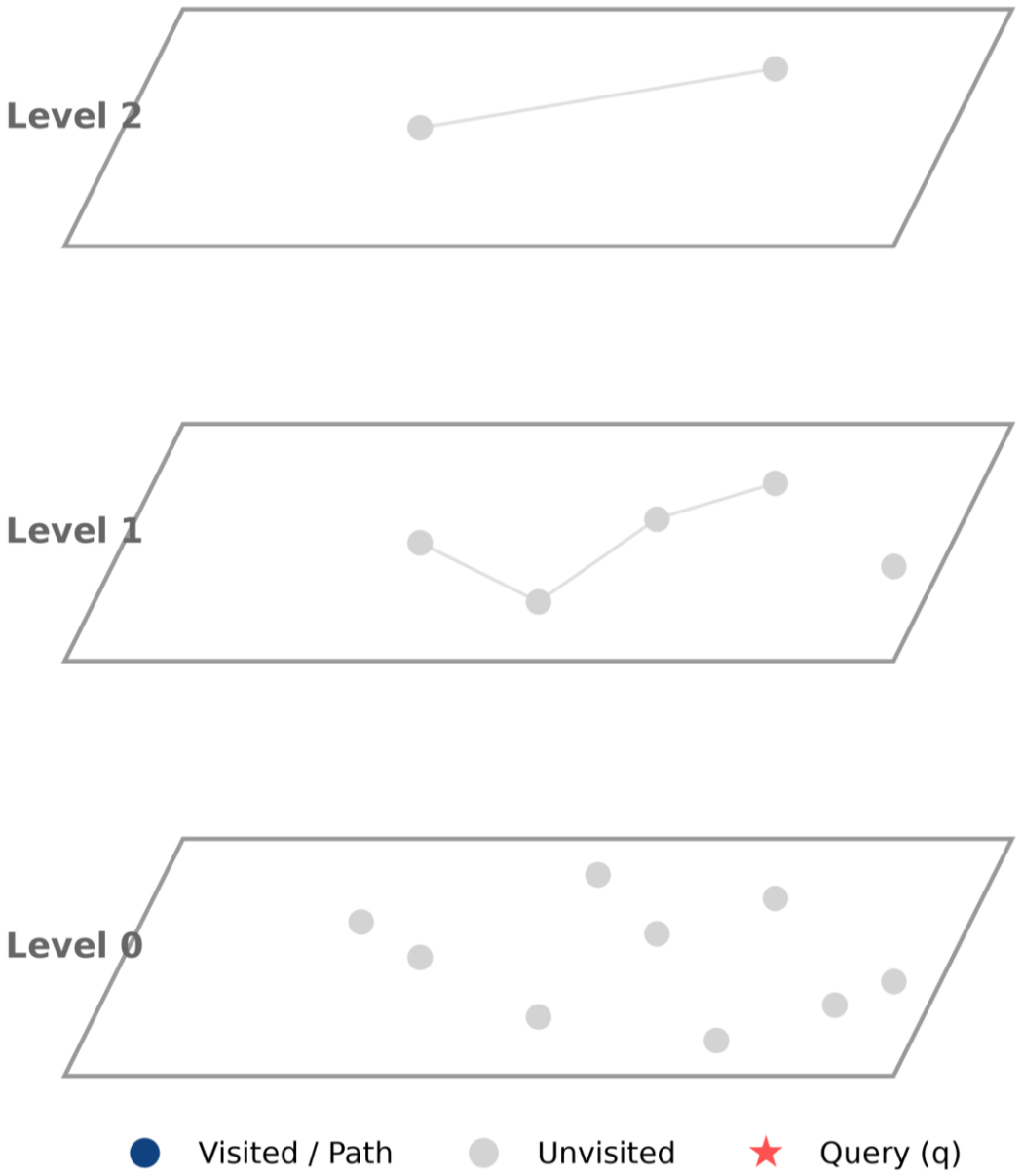
Hierarchical Navigable Small World (HNSW)

- Multi-layer proximity graph
- Greedy search from upper layers to lower layers
- Approximate nearest neighbor search without scanning every vector

UNIVERSITY OF WATERLOO

HNSW

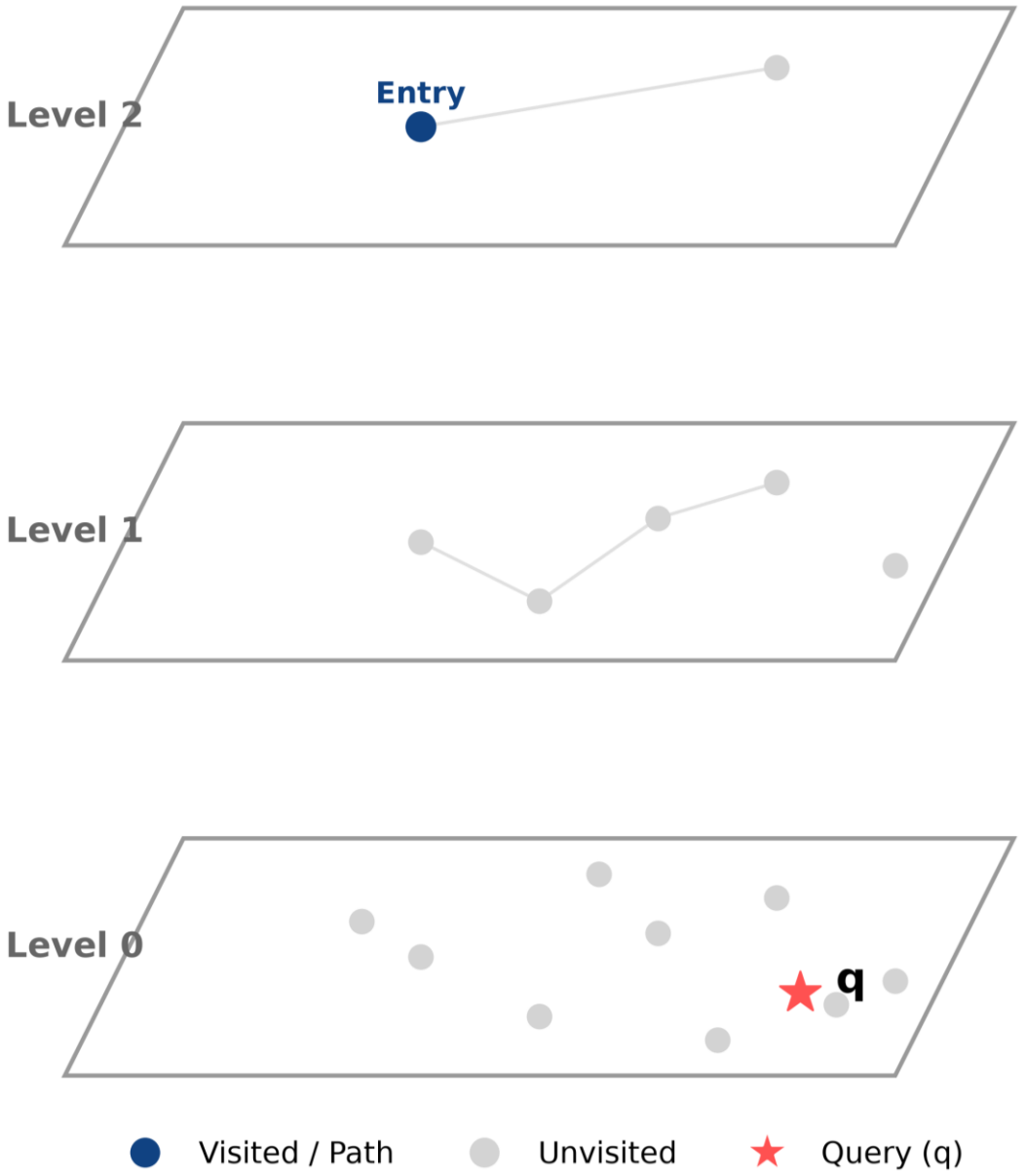
Index Construction



UNIVERSITY OF WATERLOO

HNSW

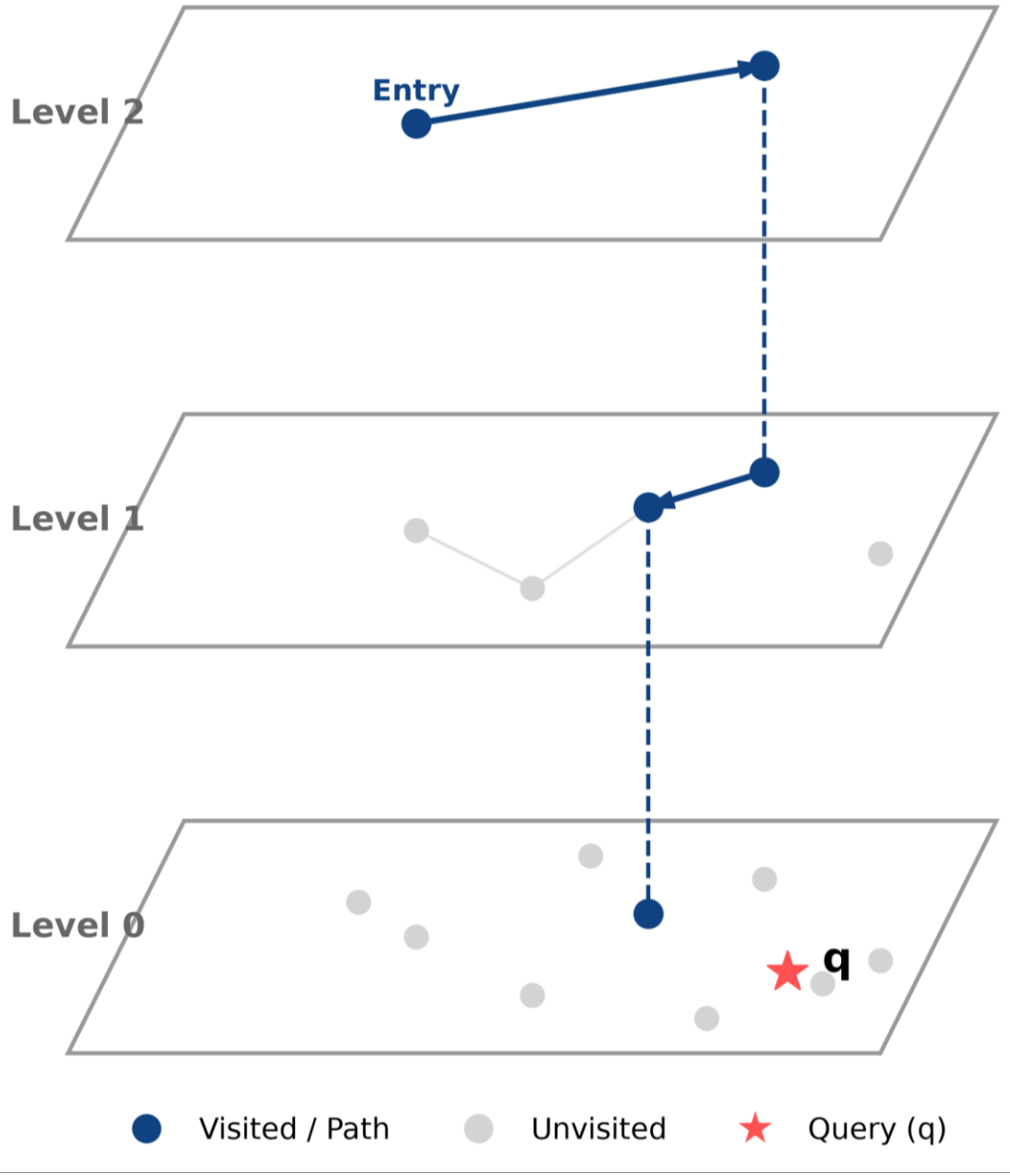
Entry Point



UNIVERSITY OF WATERLOO

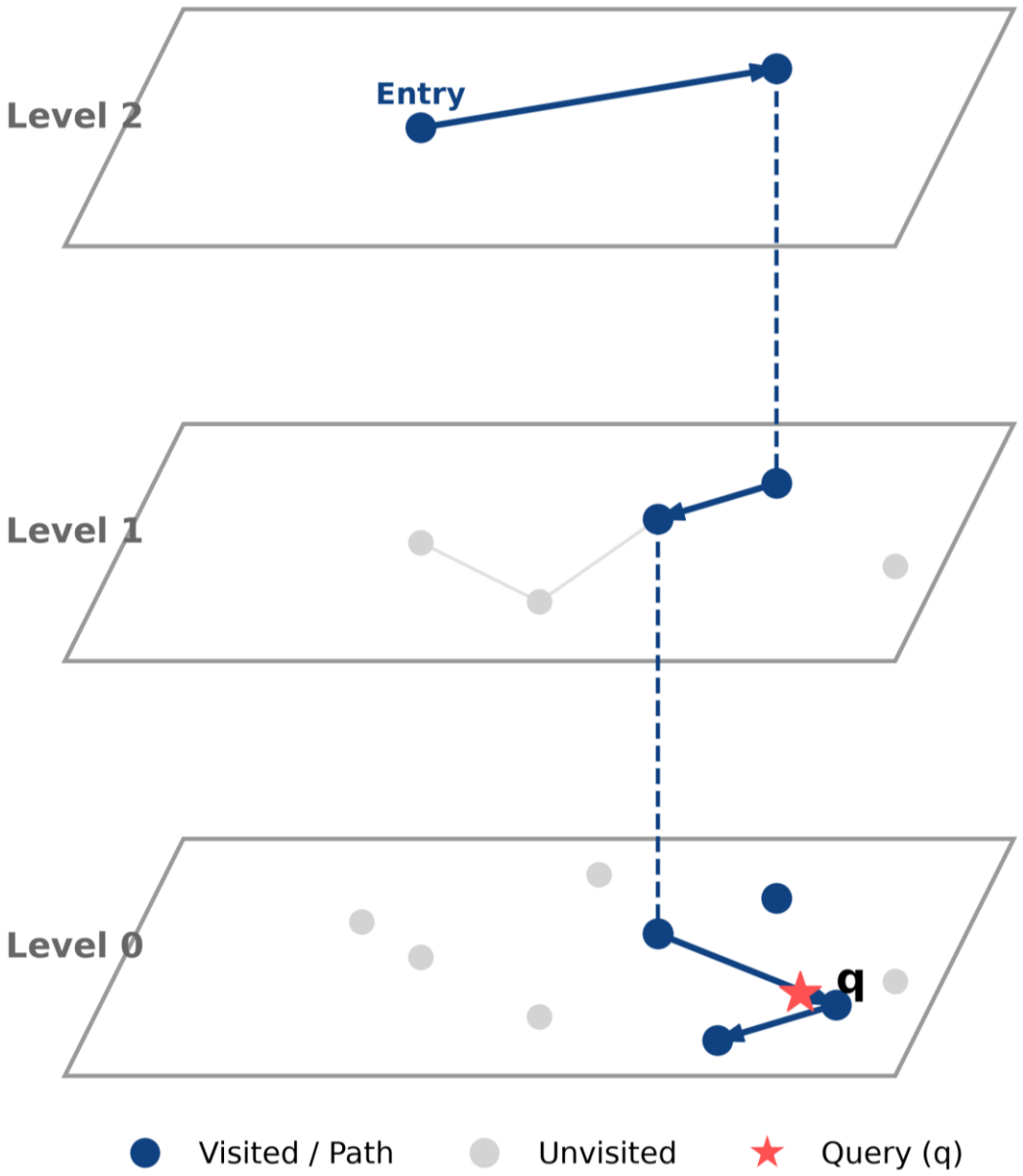
HNSW

Greedy Traversal



HNSW

Layer 0 Search



HNSW Construction: RNG Pruning

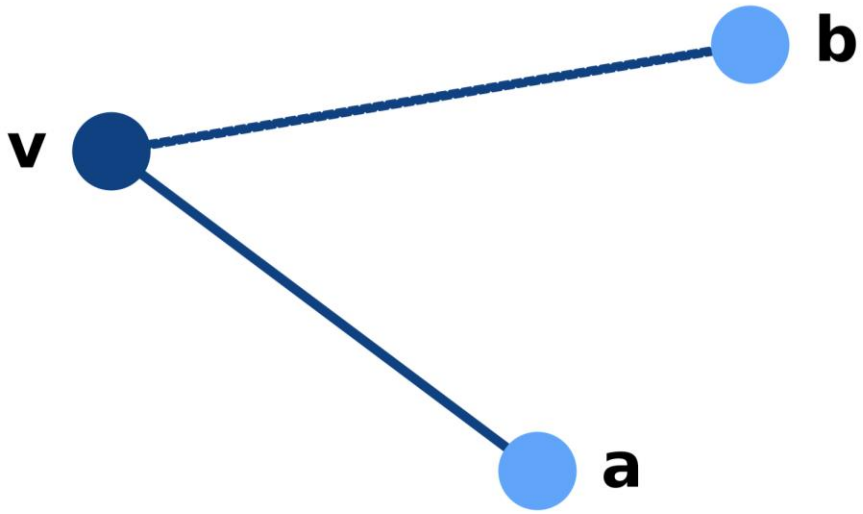
- Select up to *M* neighbors for each node
- Use RNG-based pruning to remove redundant edges
- Maintain a sparse and navigable graph

→ Avoid unnecessary connections while preserving useful search paths.

UNIVERSITY OF WATERLOO

RNG Pruning

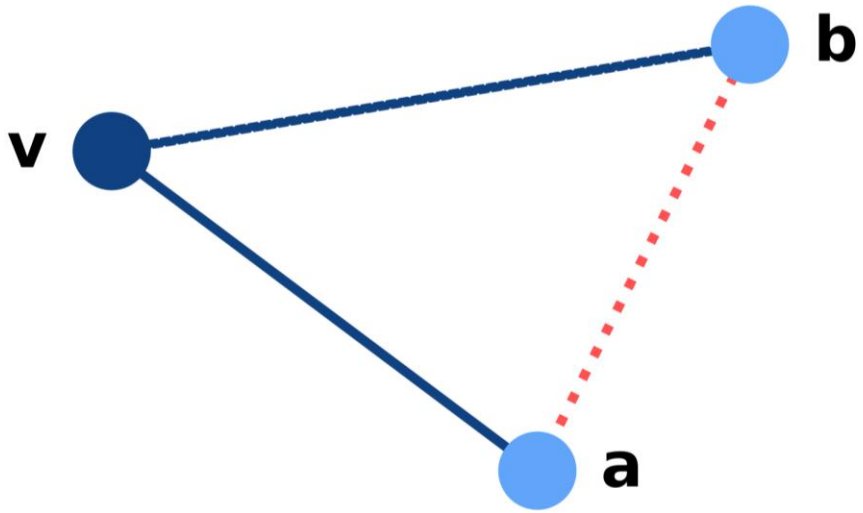
Consider the candidate edge v → b



UNIVERSITY OF WATERLOO

RNG Pruning

Node a is already selected,
and b is closer to a than to v.

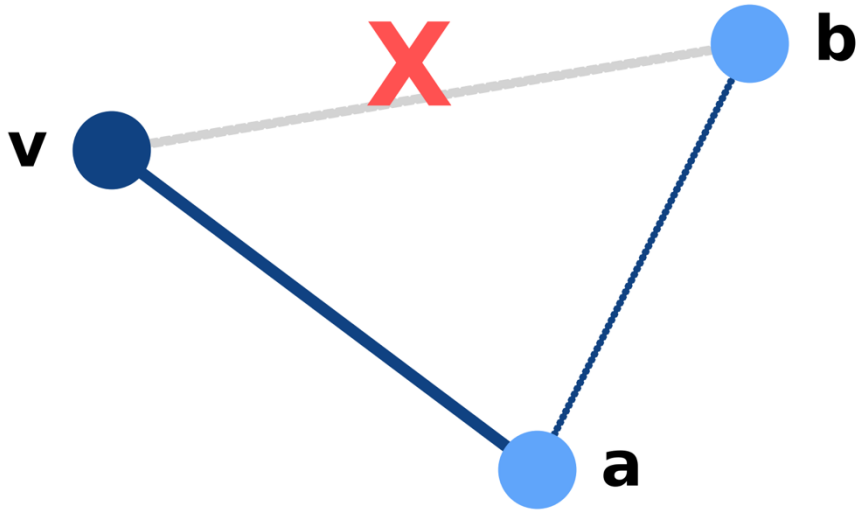


dist(a, b) < dist(v, b)

UNIVERSITY OF WATERLOO

RNG Pruning

HNSW can prune v → b because the route through a provides an alternative



UNIVERSITY OF WATERLOO

Why Standard HNSW Struggles with Filtering

- HNSW pruning does not consider future query predicates
- Filtered-out nodes may break useful search paths
- Predicate-induced subgraphs may become poorly connected

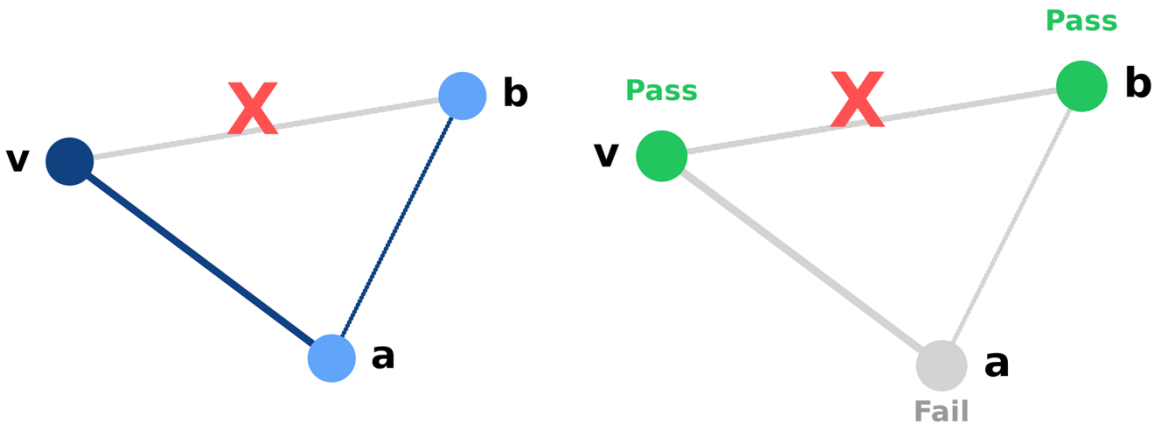
→ An edge that is redundant in the full graph may be essential in a filtered subgraph.

UNIVERSITY OF WATERLOO

Why Standard HNSW Struggles with Filtering

- HNSW pruning does not consider future query predicates
- Filtered-out nodes may break useful search paths
- Predicate-induced subgraphs may become poorly connected

→ An edge that is redundant in the full graph may be essential in a filtered subgraph.

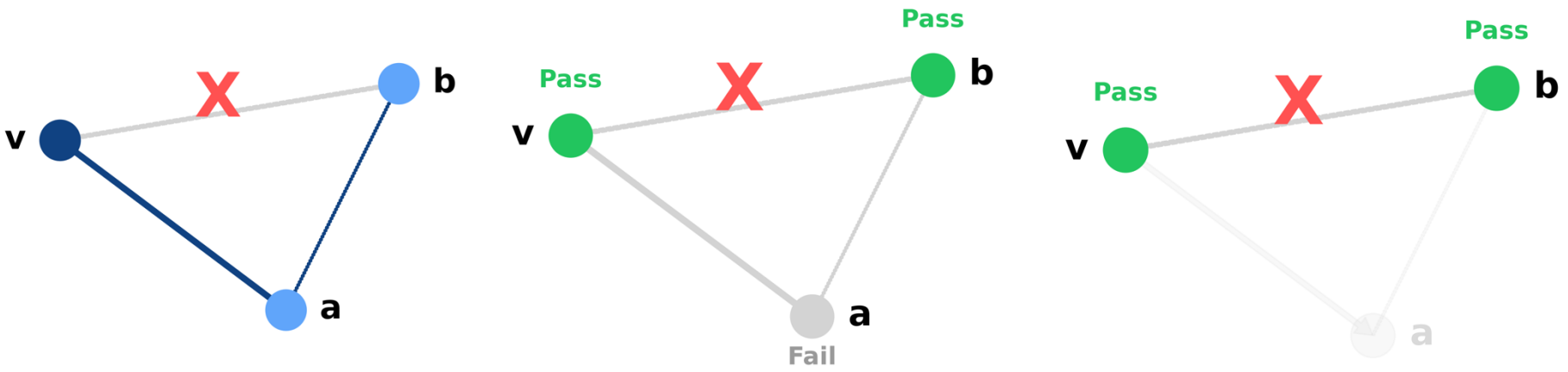


UNIVERSITY OF WATERLOO

Why Standard HNSW Struggles with Filtering

- HNSW pruning does not consider future query predicates
- Filtered-out nodes may break useful search paths
- Predicate-induced subgraphs may become poorly connected

→ An edge that is redundant in the full graph may be essential in a filtered subgraph.

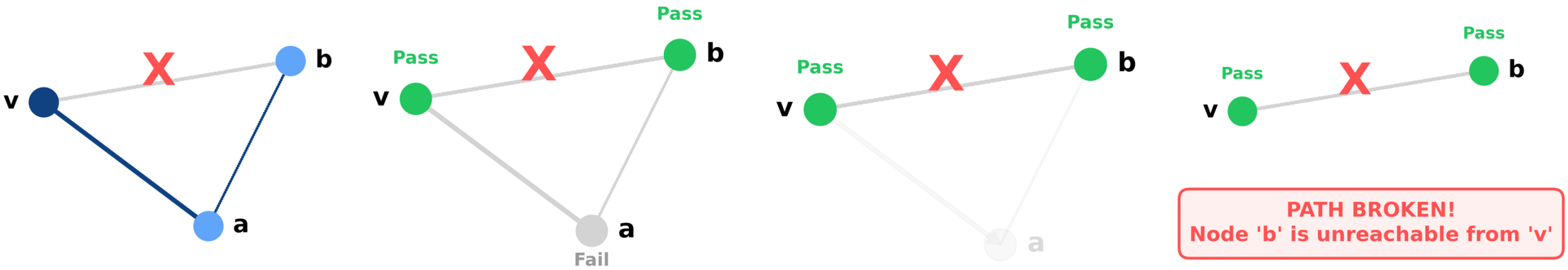


UNIVERSITY OF WATERLOO

Why Standard HNSW Struggles with Filtering

- HNSW pruning does not consider future query predicates
- Filtered-out nodes may break useful search paths
- Predicate-induced subgraphs may become poorly connected

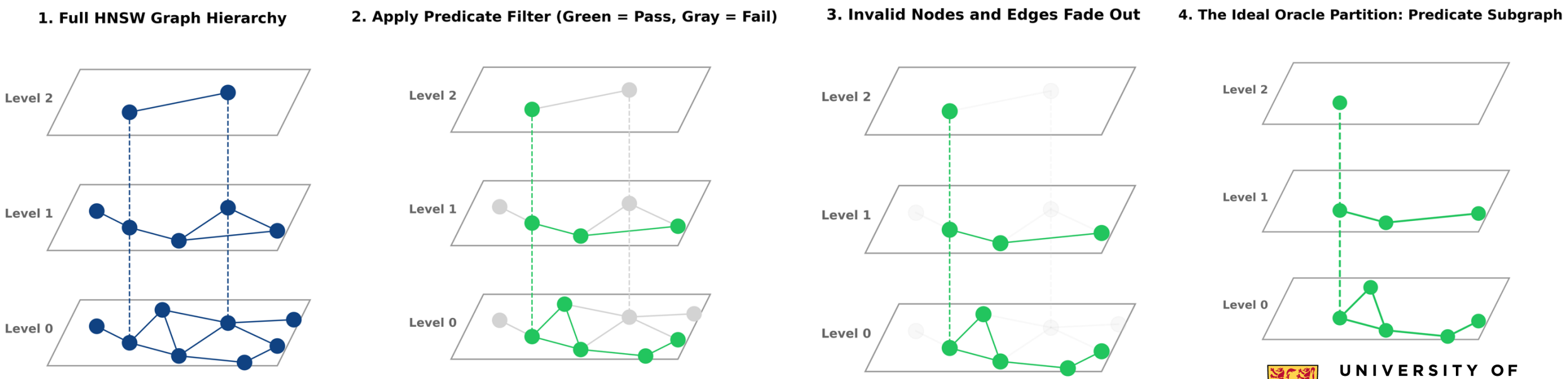
→ An edge that is redundant in the full graph may be essential in a filtered subgraph.



UNIVERSITY OF WATERLOO

The Ideal Solution: Predicate Subgraphs

- Oracle Partition Index: Build a separate HNSW index over vectors satisfying each predicate.
- Search directly within the predicate-specific graph.
- Ideal search complexity: $O(\log(sn) + K)$.
- Challenge: Predicates can be arbitrary, numerous, or unknown at construction time.



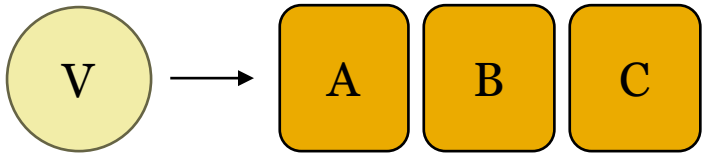
UNIVERSITY OF WATERLOO

ACORN-γ: Neighbor List Expansion

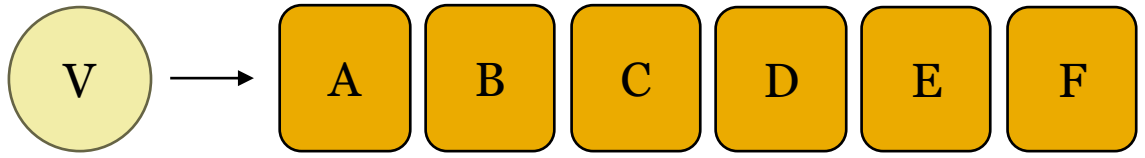
- HNSW: Collect up to M candidate neighbors.
- ACORN-γ: Collect up to $M * \gamma$ candidate neighbors.
- Expansion is independent of structured predicates.
- A denser graph provides more potential edges after filtering.

Example: $M = 3, \gamma = 2$ $Candidate\ List\ Size = M * \gamma = 6$

Standard HNSW



ACORN-γ



UNIVERSITY OF WATERLOO

ACORN-γ: Compression

- **Step1:** Preserve the first M_β closest candidate neighbors.
- **Step2:** Compress the remaining candidates using 2-hop reachability.
- **Step3:** Prune a candidate if it is already covered by a retained neighbor.
- **Step4:** Recover pruned candidates through 2-hop expansion during search.

Keep the first M_β candidates
H = {}

For each remaining candidate c:
  *if |H| + number_of_kept_edges > M * γ:*
    break

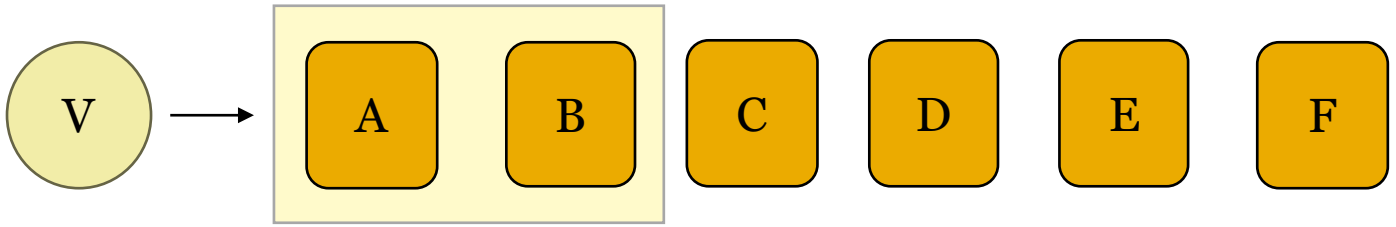
  if c in H:
    Prune c
  else:
    Keep c
    H = H ∪ Neighbors(c)

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Step1: Preserve the first M_β closest candidate neighbors.

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



Always keep

$M_\beta = 2$

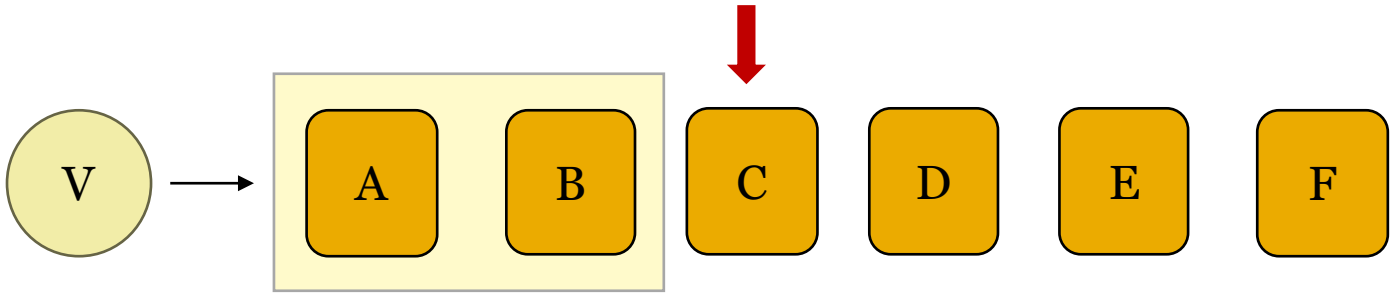
$\boldsymbol{H} = \emptyset$

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Step1: Preserve the first M_β closest candidate neighbors.

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



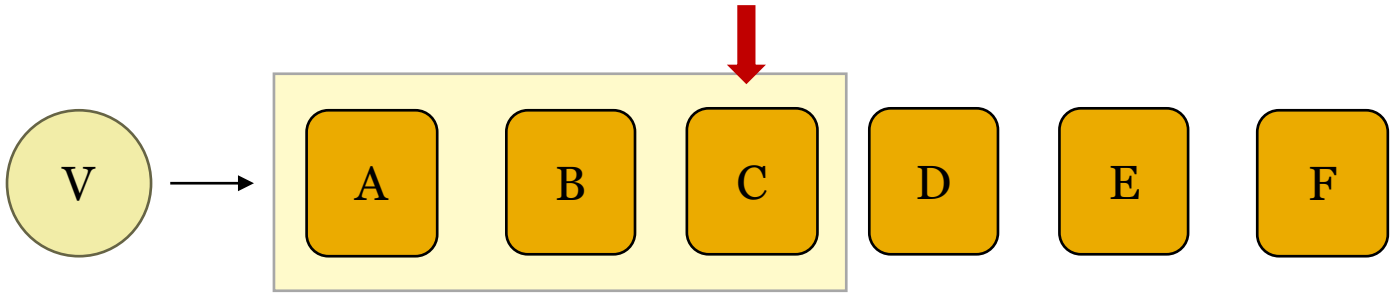
$\boldsymbol{H} = \emptyset$

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Step1: Preserve the first M_β closest candidate neighbors.

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



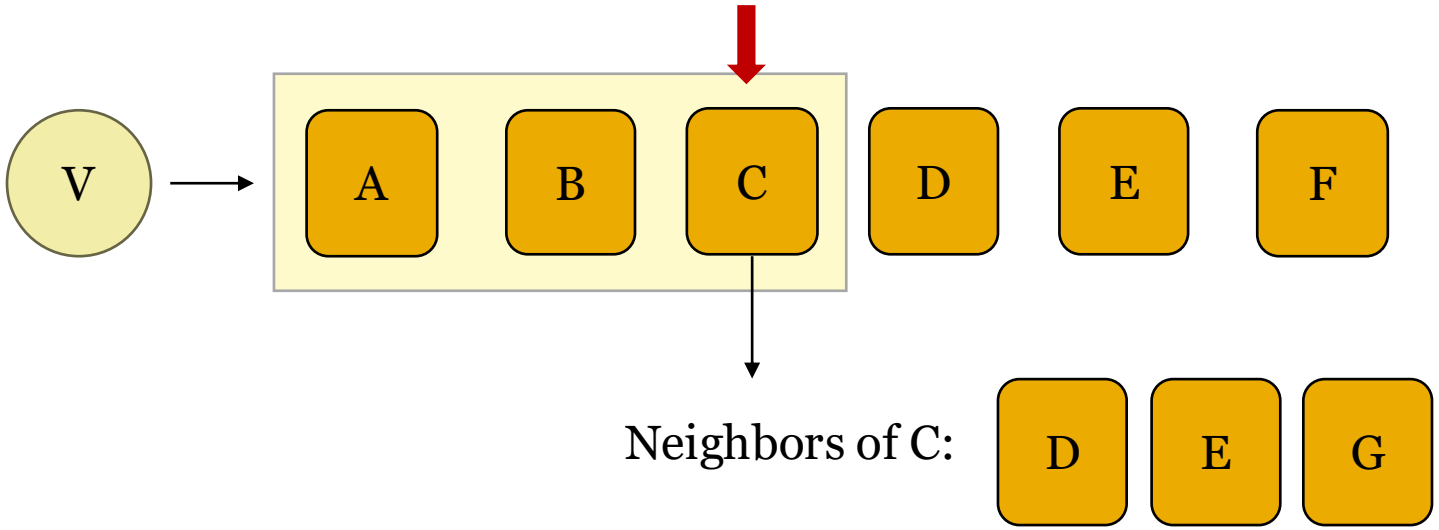
$\boldsymbol{H} = \emptyset$

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Step2: Compress the remaining candidates using 2-hop reachability

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



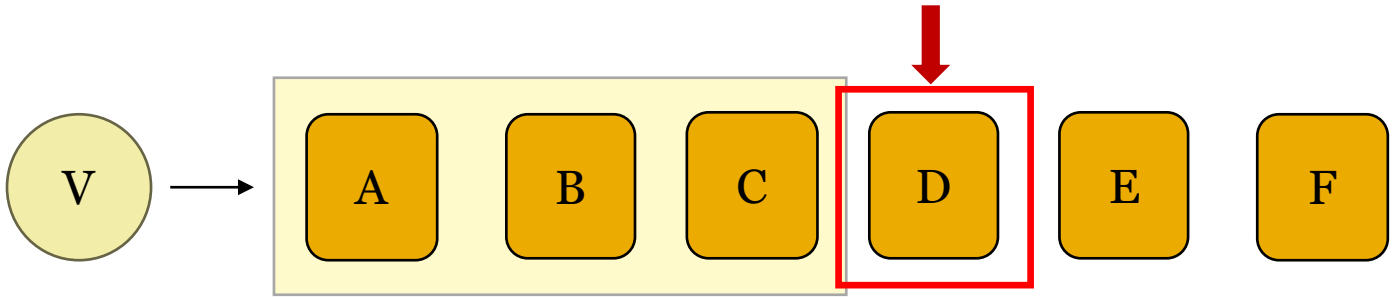
$\boldsymbol{H = \{D, E, G\}}$

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Step2: Compress the remaining candidates using 2-hop reachability

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



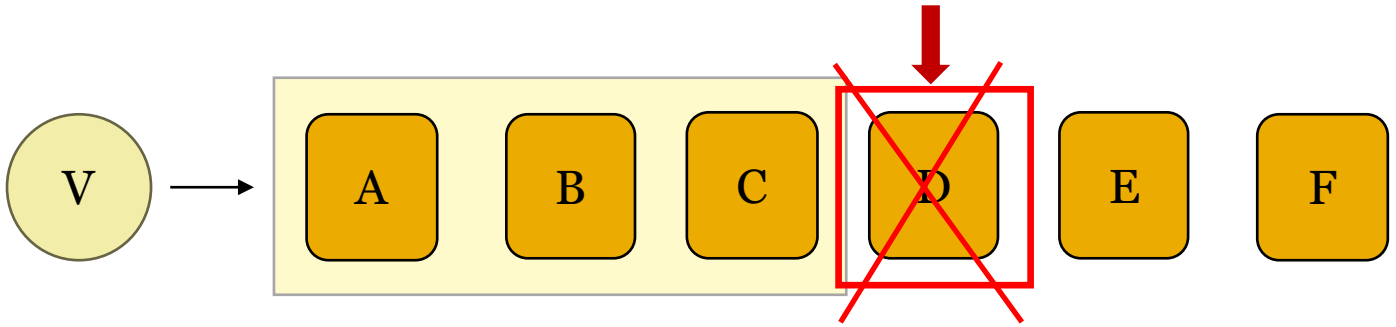
$$\boldsymbol{H = \{D, E, G\}}$$

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Step3: Prune a candidate if it is already covered by a retained neighbor.

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



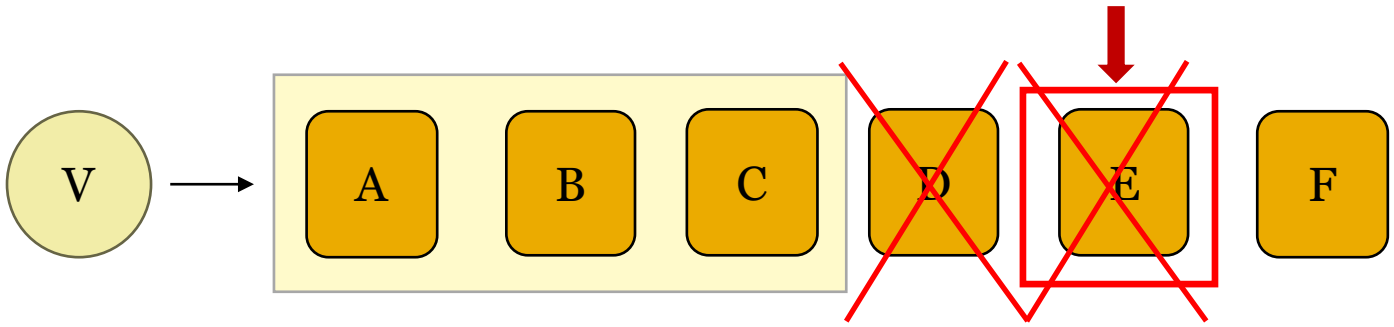
$$\boldsymbol{H = \{D, E, G\}}$$

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Step3: Prune a candidate if it is already covered by a retained neighbor.

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



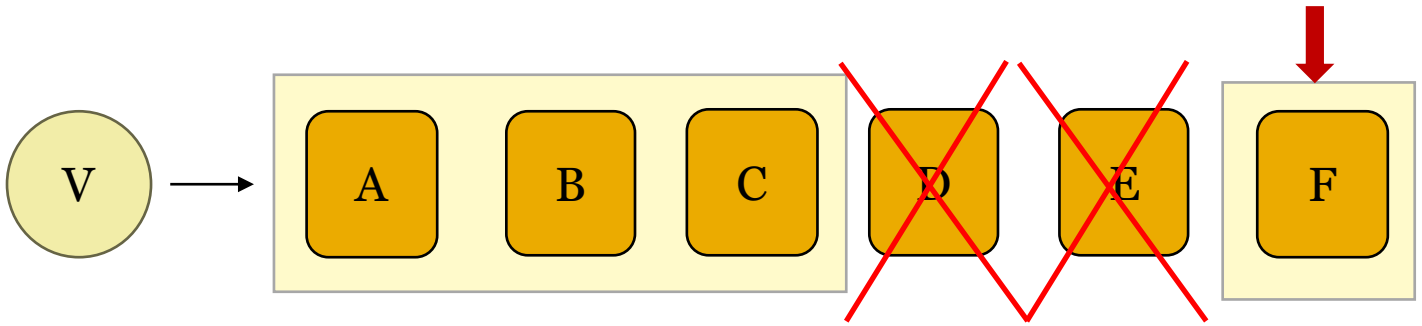
$$\boldsymbol{H = \{D, E, G\}}$$

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Step3: Prune a candidate if it is already covered by a retained neighbor.

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



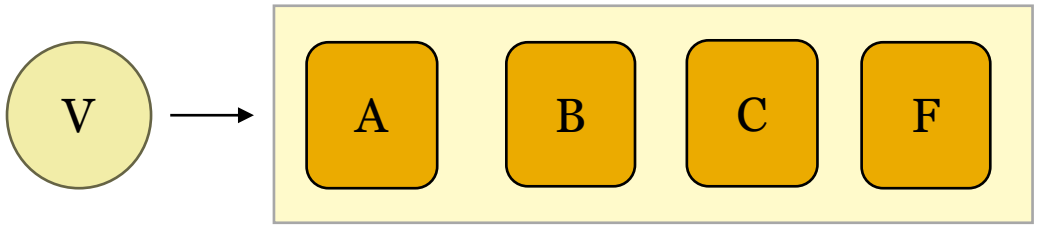
H ← H ∪ Neighbors(F)

$\boldsymbol{H = \{D, E, G\}}$

UNIVERSITY OF WATERLOO

ACORN-γ: Compression

Example: $M = 3, \quad \gamma = 2, \quad M_\beta = 2$



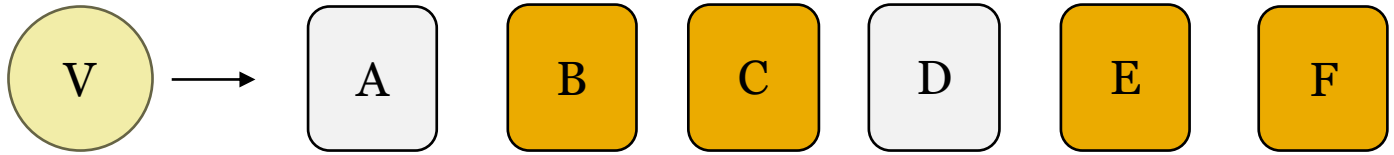
UNIVERSITY OF WATERLOO

ACORN-γ Search: Predicate-Based Filtering

- Access a node's expanded, uncompressed neighbor list.
- Evaluate the query predicate for each neighbor.
- Discard neighbors that fail the predicate.
- Use at most the first M matching neighbors for graph traversal.

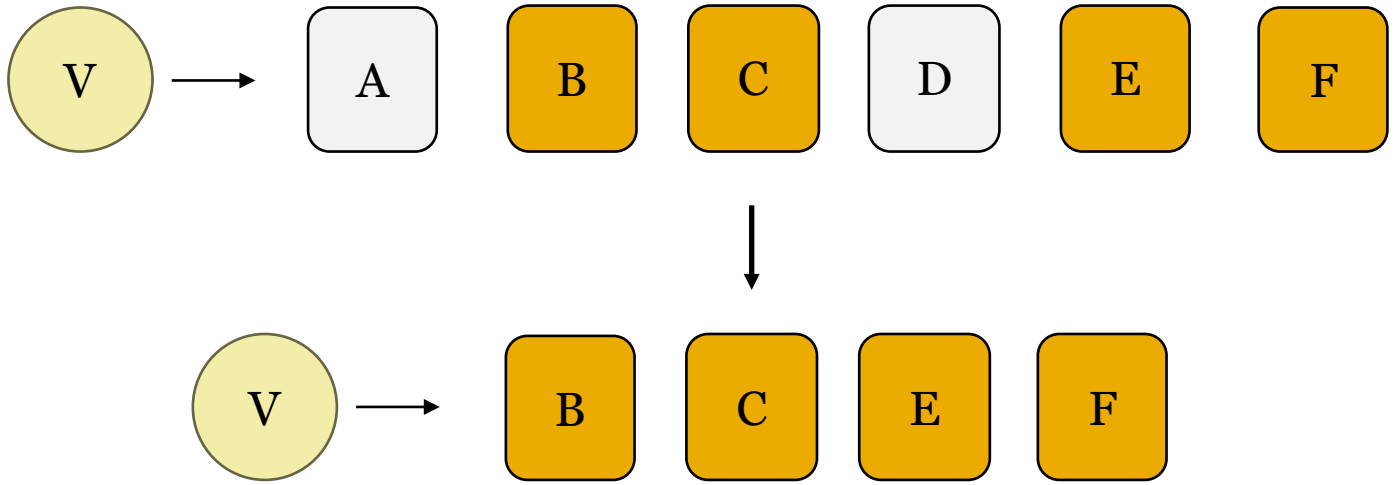
UNIVERSITY OF WATERLOO

ACORN-γ Search: Predicate-Based Filtering



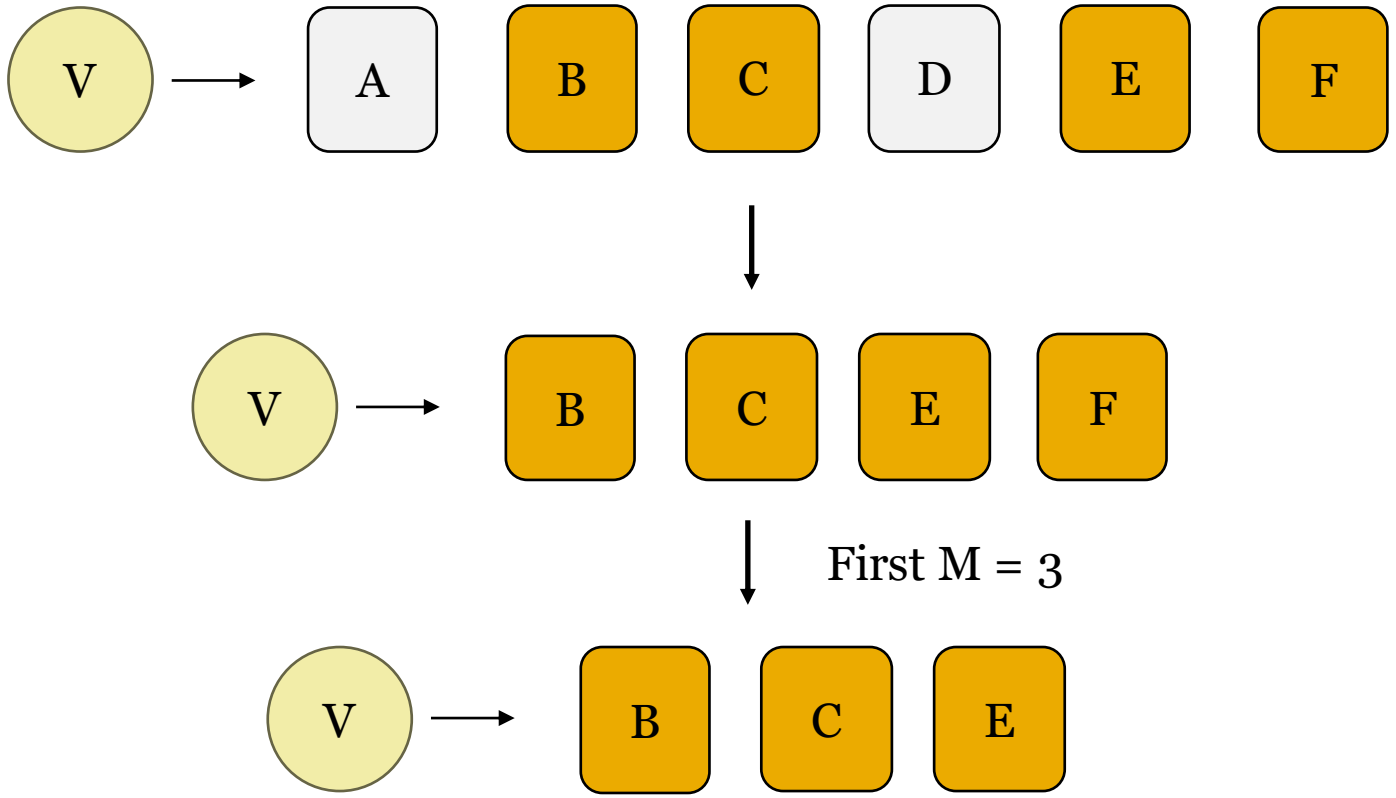
UNIVERSITY OF WATERLOO

ACORN-γ Search: Predicate-Based Filtering



UNIVERSITY OF WATERLOO

ACORN-γ Search: Predicate-Based Filtering



UNIVERSITY OF WATERLOO

ACORN-γ Search: Compression-Based Heuristic

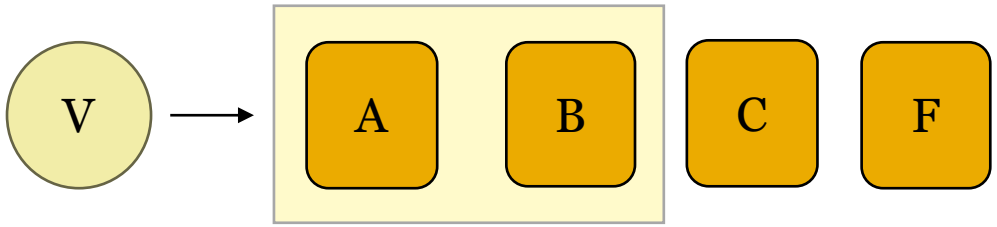
- Phase 1: Directly inspect and filter the first M_β neighbors.
- Phase 2: Expand the remaining stored neighbors to include their neighbors.
- Filter the combined candidates using the query predicate.
- Return at most M matching neighbors.

UNIVERSITY OF WATERLOO

ACORN-γ Search: Compression-Based Heuristic

Phase 1: Directly inspect and filter the first M_β neighbors.

$$M = 3, \quad \gamma = 2, \quad M_\beta = 2$$



Directly inspect

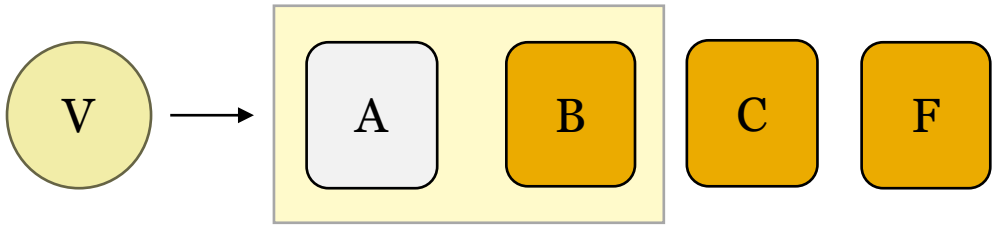
$M_\beta = 2$

UNIVERSITY OF WATERLOO

ACORN-γ Search: Compression-Based Heuristic

Phase 1: Directly inspect and filter the first M_β neighbors.

$$M = 3, \quad \gamma = 2, \quad M_\beta = 2$$



Directly inspect

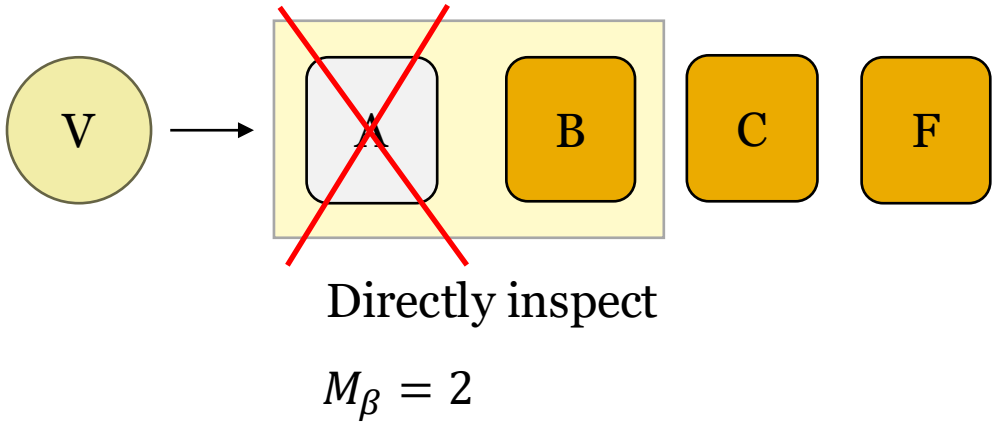
$M_\beta = 2$

UNIVERSITY OF WATERLOO

ACORN-γ Search: Compression-Based Heuristic

Phase 1: Directly inspect and filter the first M_β neighbors.

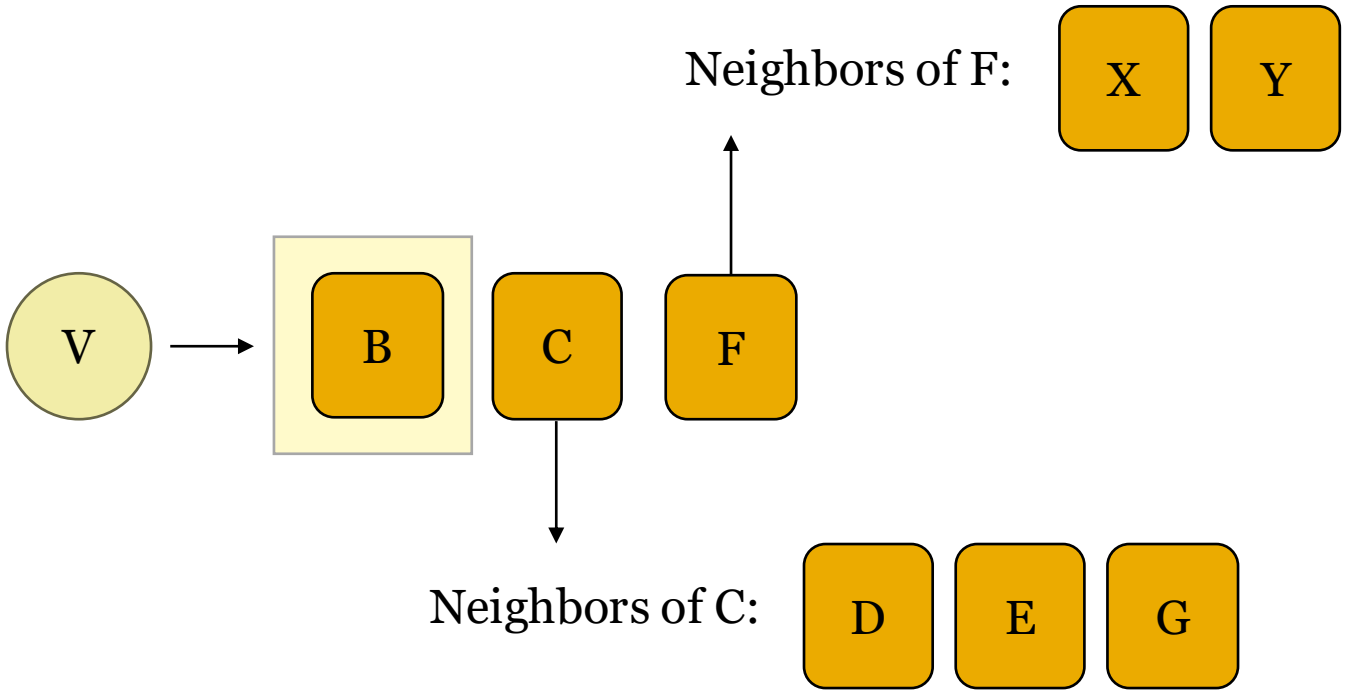
$$M = 3, \quad \gamma = 2, \quad M_\beta = 2$$



UNIVERSITY OF WATERLOO

ACORN-γ Search: Compression-Based Heuristic

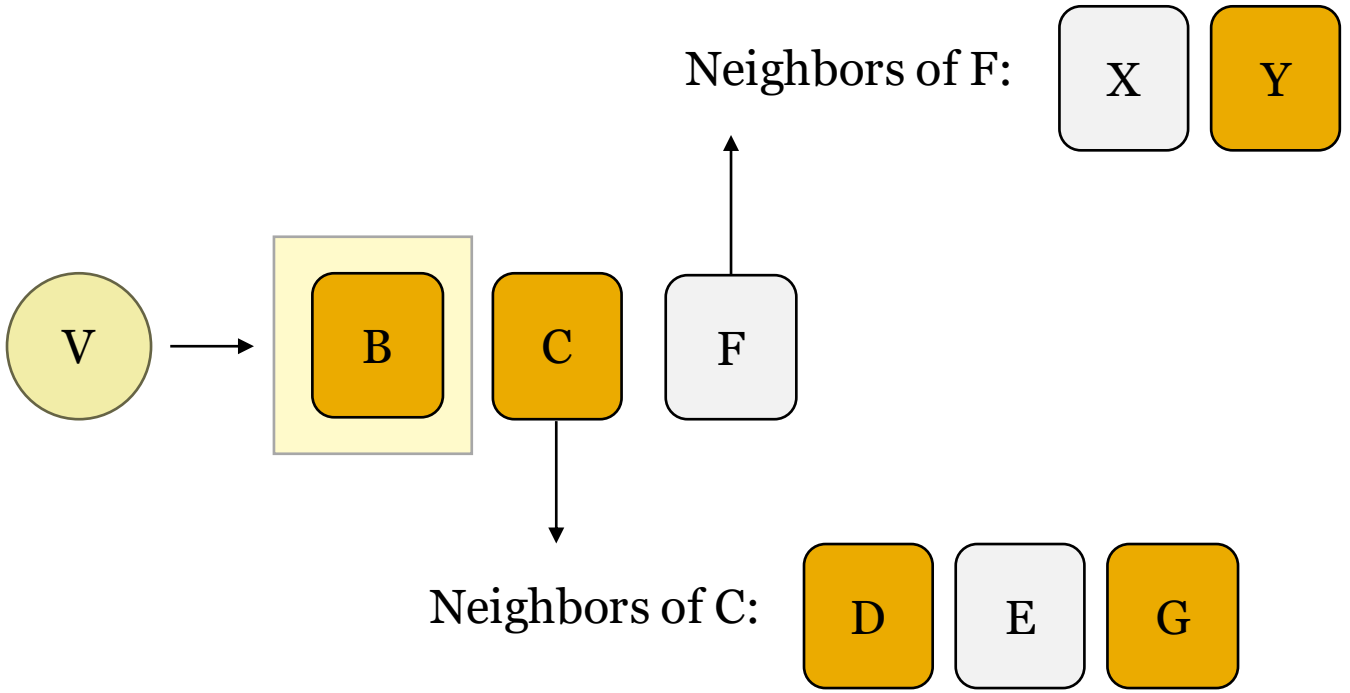
Phase 2: Expand the remaining stored neighbors to include their neighbors.



UNIVERSITY OF WATERLOO

ACORN-γ Search: Compression-Based Heuristic

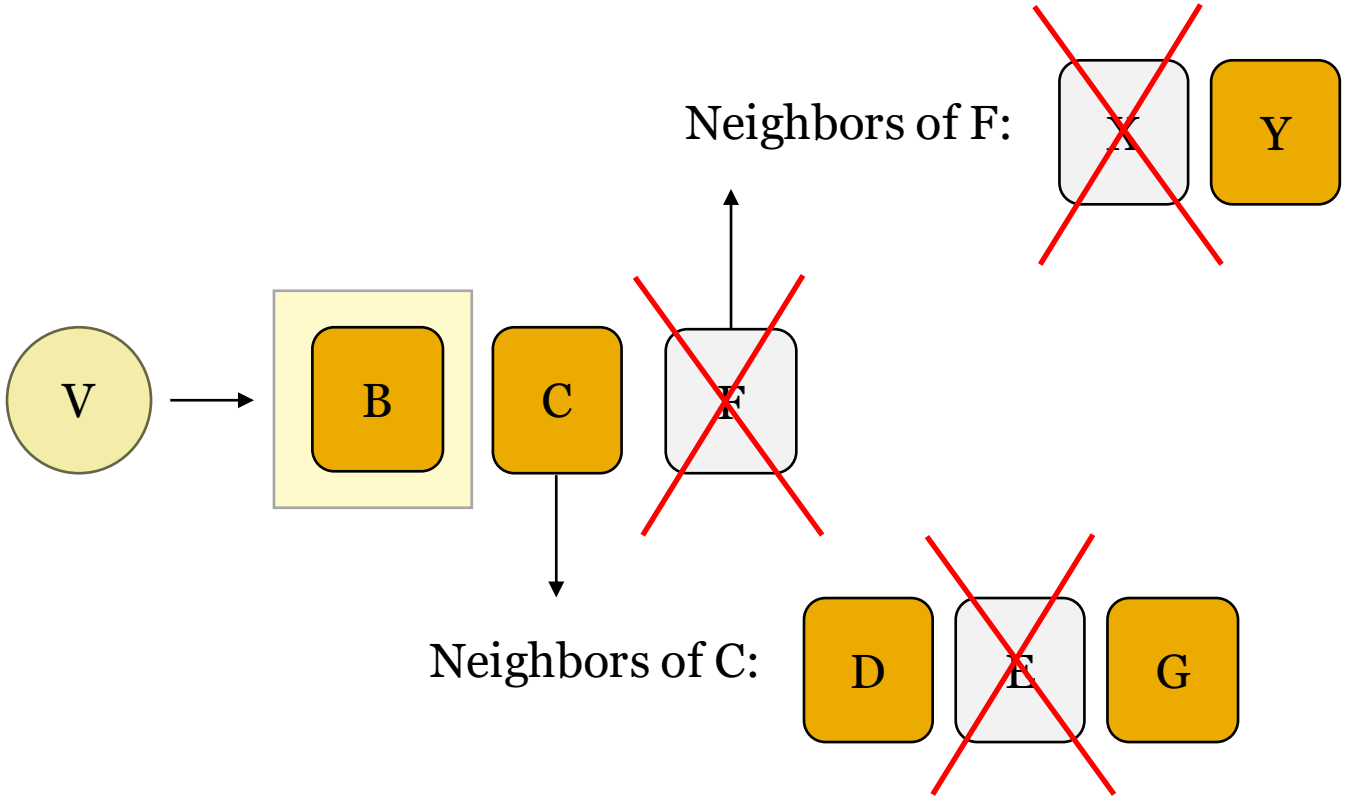
Phase 2: Expand the remaining stored neighbors to include their neighbors.



UNIVERSITY OF WATERLOO

ACORN-γ Search: Compression-Based Heuristic

Phase 2: Expand the remaining stored neighbors to include their neighbors.

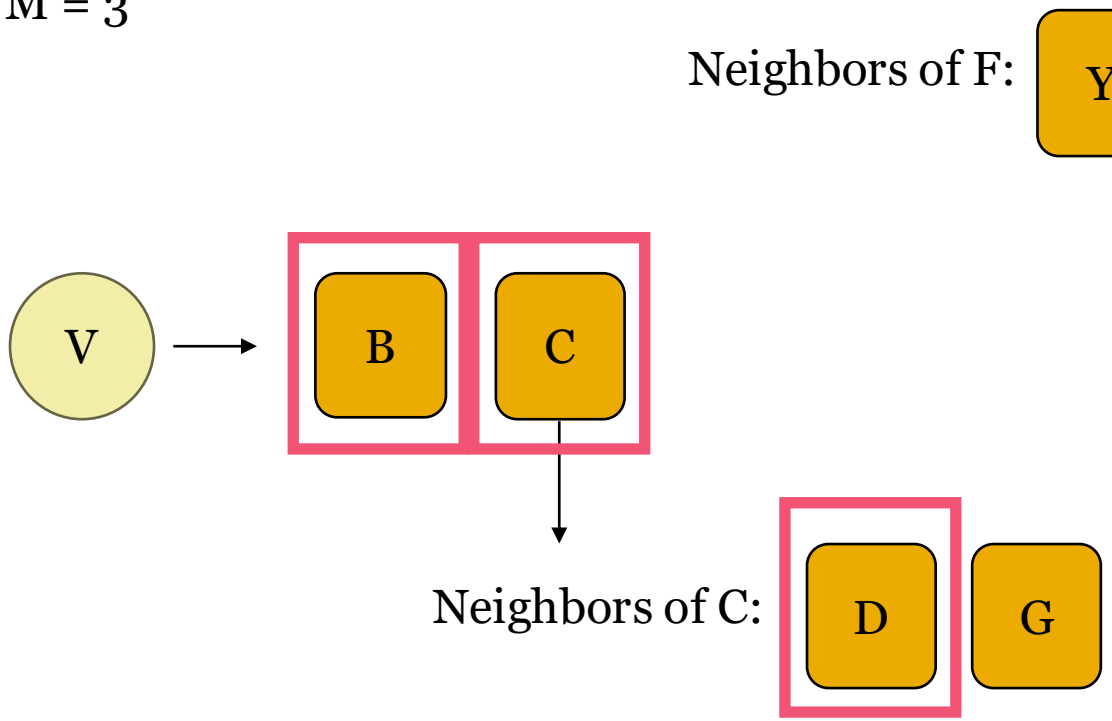


UNIVERSITY OF WATERLOO

ACORN-γ Search: Compression-Based Heuristic

Phase 2: Expand the remaining stored neighbors to include their neighbors.

Return First M = 3



UNIVERSITY OF WATERLOO

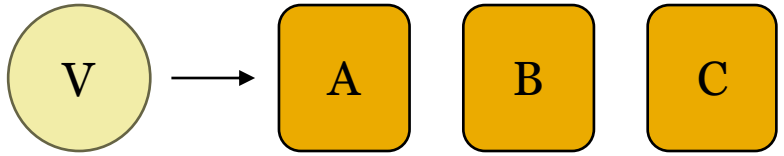
ACORN-1: Search-Time Neighbor Expansion

- Construct a graph without neighbor-list expansion or RNG pruning.
- Equivalent construction parameters: $\gamma = 1, M_\beta = M$.
- At search time, expand all one-hop and two-hop neighbors.
- Apply predicate filtering and retain up to M neighbors.
- Lower construction overhead, at the cost of more work during search.

UNIVERSITY OF WATERLOO

ACORN-1: Search-Time Neighbor Expansion

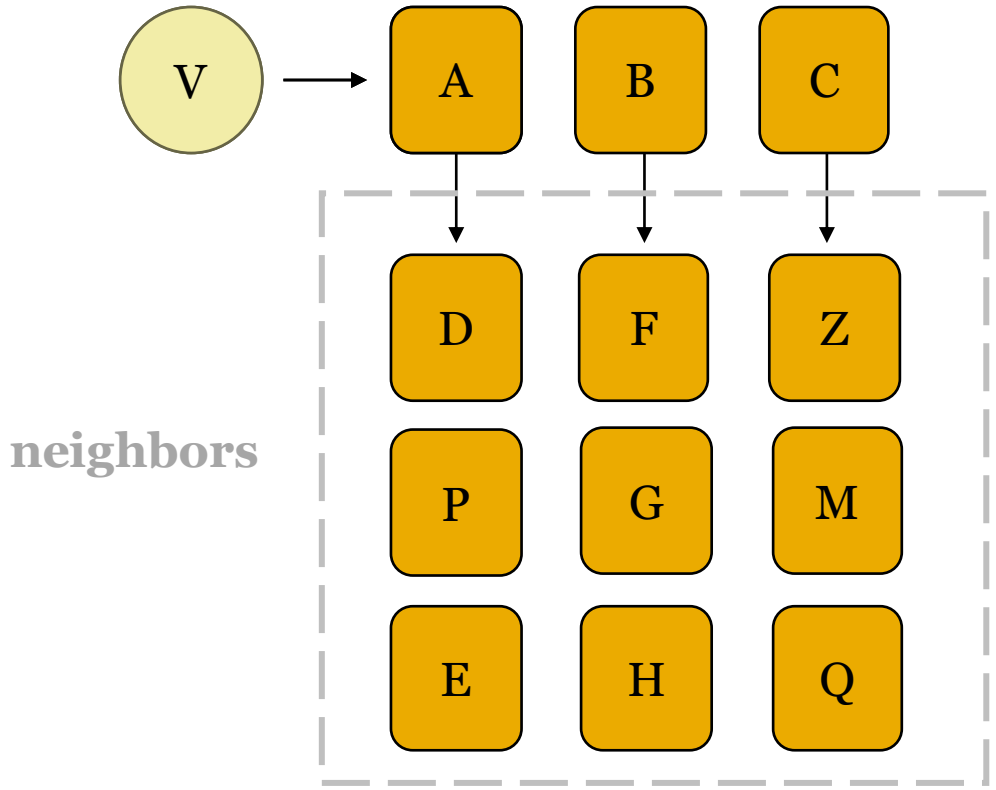
$M = 3, \quad \gamma = 1, \quad M_\beta = M$



UNIVERSITY OF WATERLOO

ACORN-1: Search-Time Neighbor Expansion

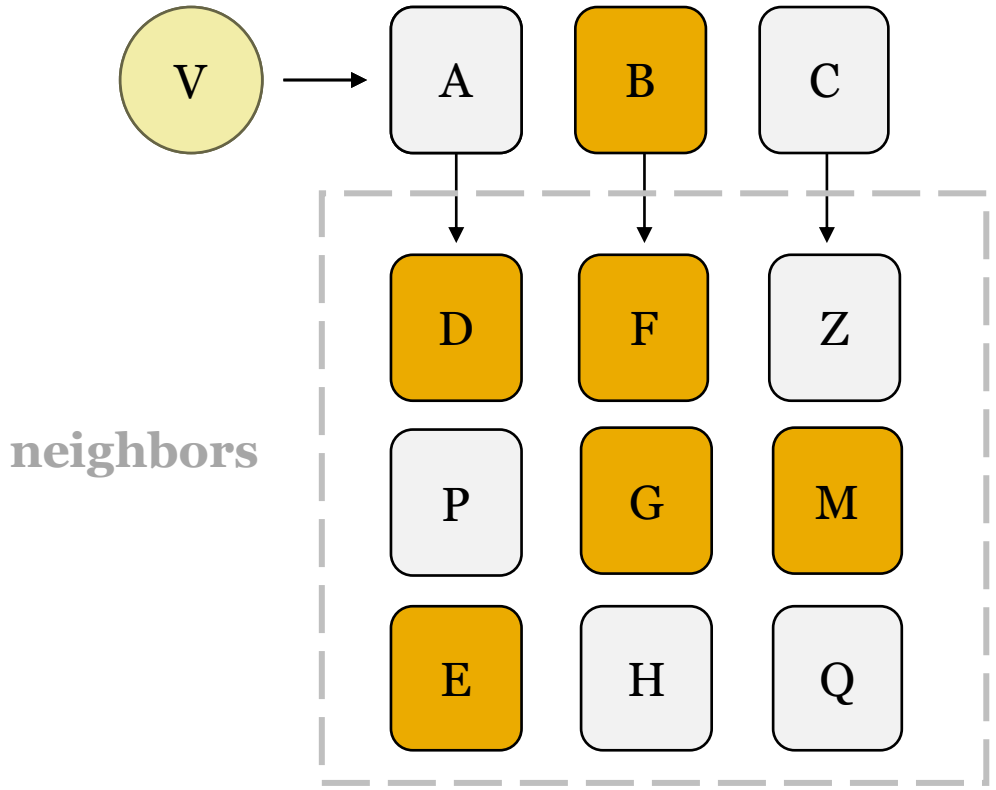
$M = 3, \quad \gamma = 1, \quad M_\beta = M$



UNIVERSITY OF WATERLOO

ACORN-1: Search-Time Neighbor Expansion

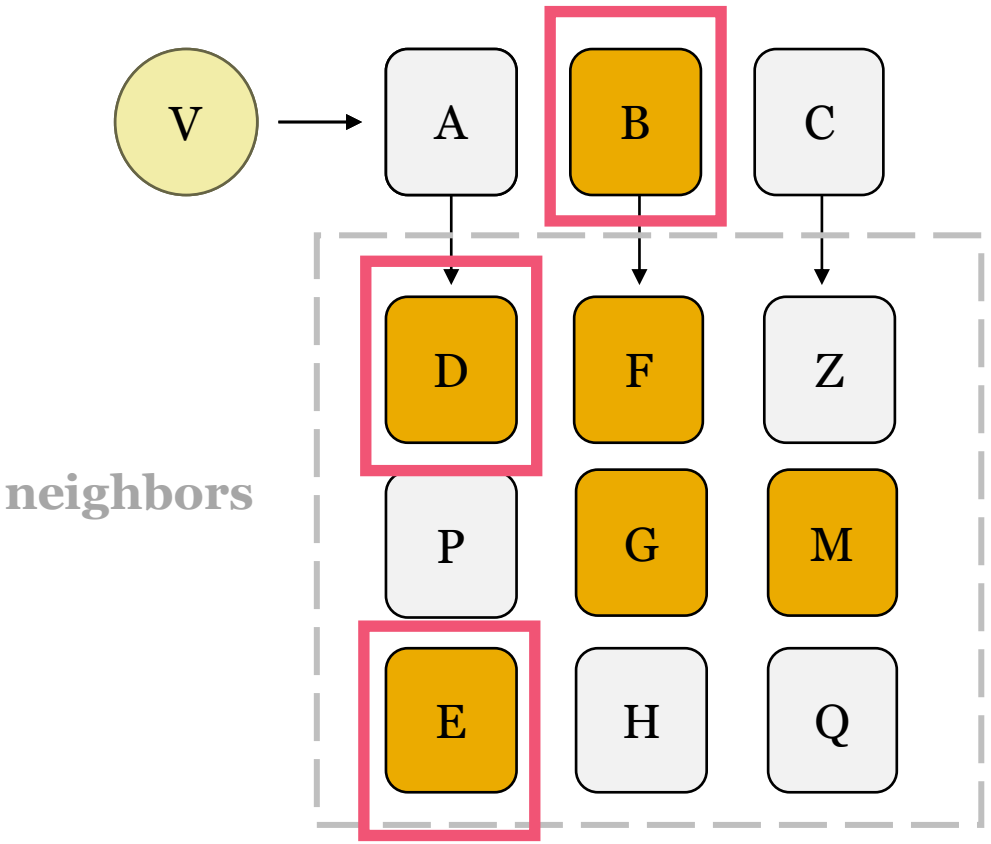
$M = 3, \quad \gamma = 1, \quad M_\beta = M$



UNIVERSITY OF WATERLOO

ACORN-1: Search-Time Neighbor Expansion

$M = 3, \quad \gamma = 1, \quad M_\beta = M$



Illustrative candidate order:
B → D → E → F → G → M

UNIVERSITY OF WATERLOO

ACORN-γ vs. ACORN-1

Dimension	ACORN-γ	ACORN-1
Neighbor expansion	Construction	Search
Graph construction	Expanded graph + Level-0 compression	Unpruned, lighter
Search lookup	Filter + partial 2-hop	Full 1-hop + 2-hop
Search efficiency	Generally higher	Generally lower
Time-to-index	Generally higher	Generally lower
Best suited for	Search-heavy workloads	Construction-sensitive workloads

ACORN-γ: Pay more upfront. ACORN-1: Pay more at query time.

UNIVERSITY OF WATERLOO

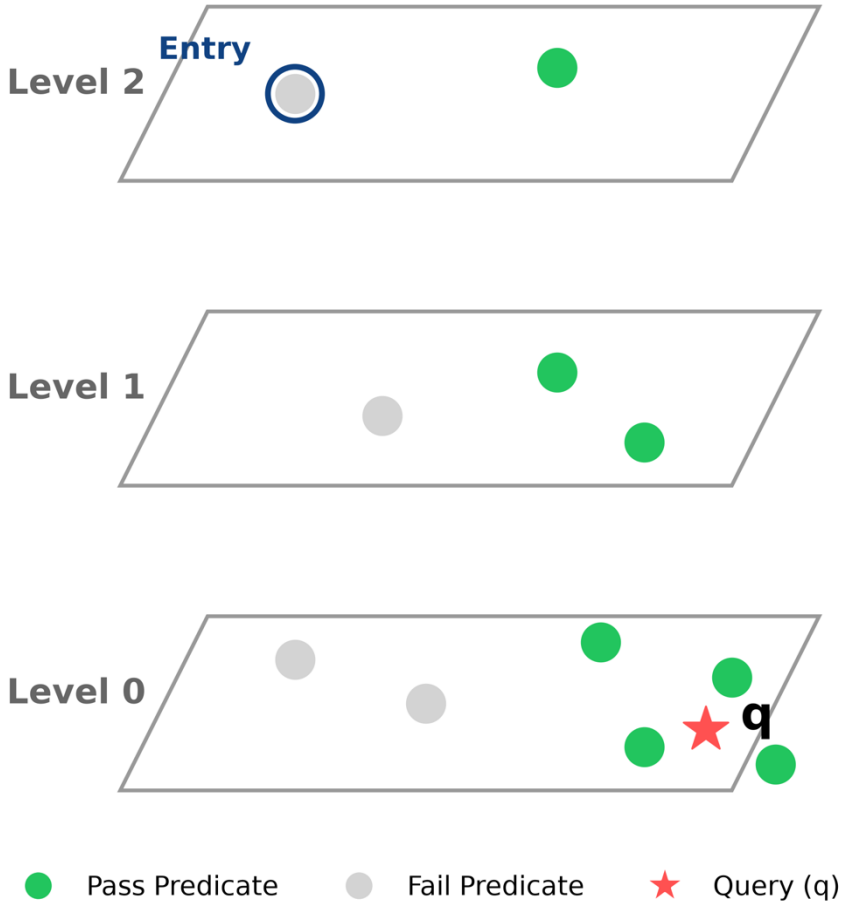
Putting It Together: ACORN Search

- Start from the graph's entry point.
- Find matching nodes and navigate into the predicate subgraph.
- Use predicate-aware neighbor lookup at each visited node.
- Maintain the best candidate results using greedy ANN search.
- Return the approximate Top-K matching vectors.

UNIVERSITY OF WATERLOO

Putting It Together: ACORN Search

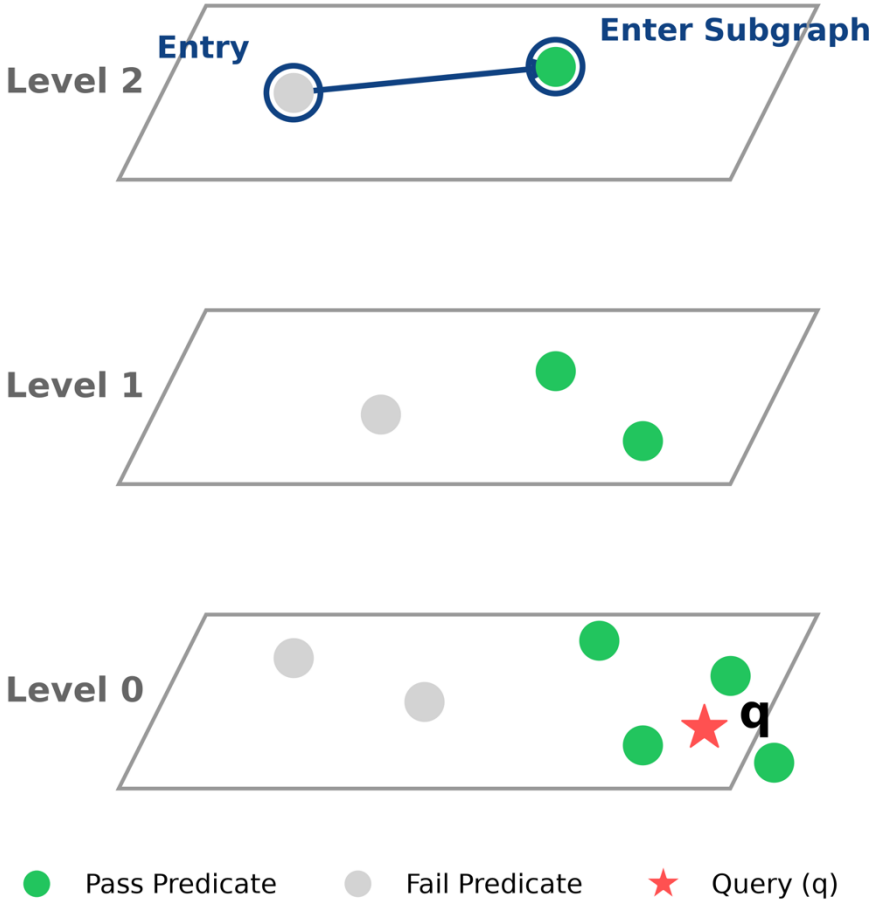
Start from Top-Level Entry Point



UNIVERSITY OF WATERLOO

Putting It Together: ACORN Search

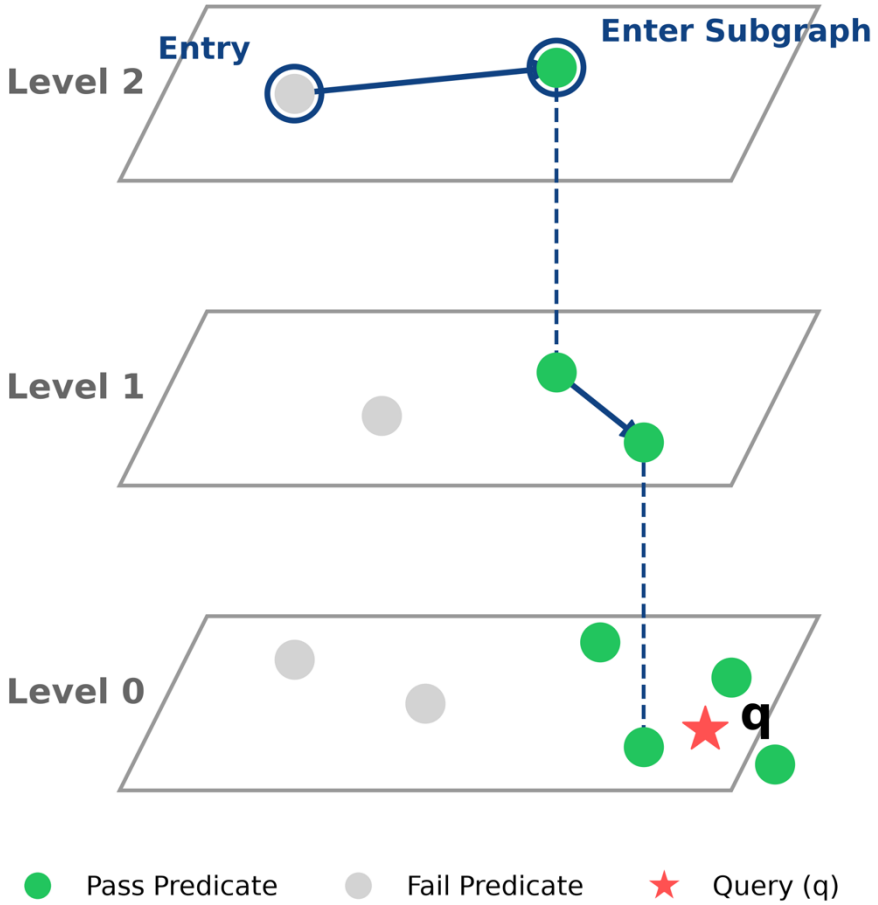
Find First Valid Node & Enter Predicate Subgraph



UNIVERSITY OF WATERLOO

Putting It Together: ACORN Search

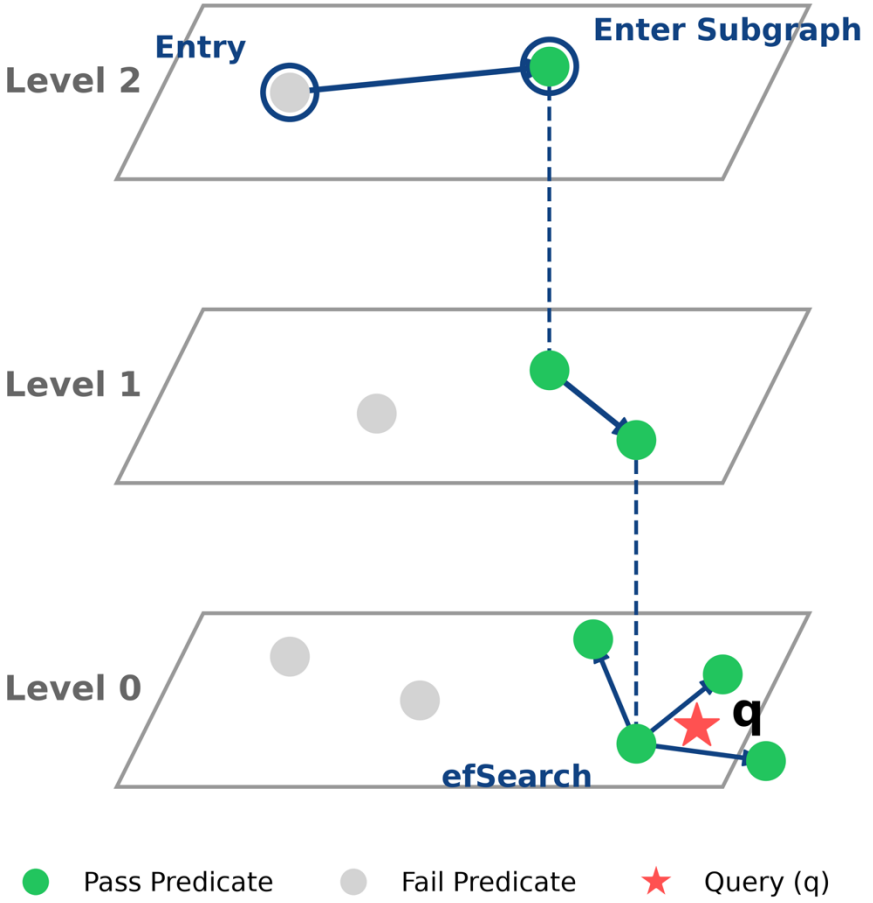
Greedy Traversal over Matching Nodes



UNIVERSITY OF WATERLOO

Putting It Together: ACORN Search

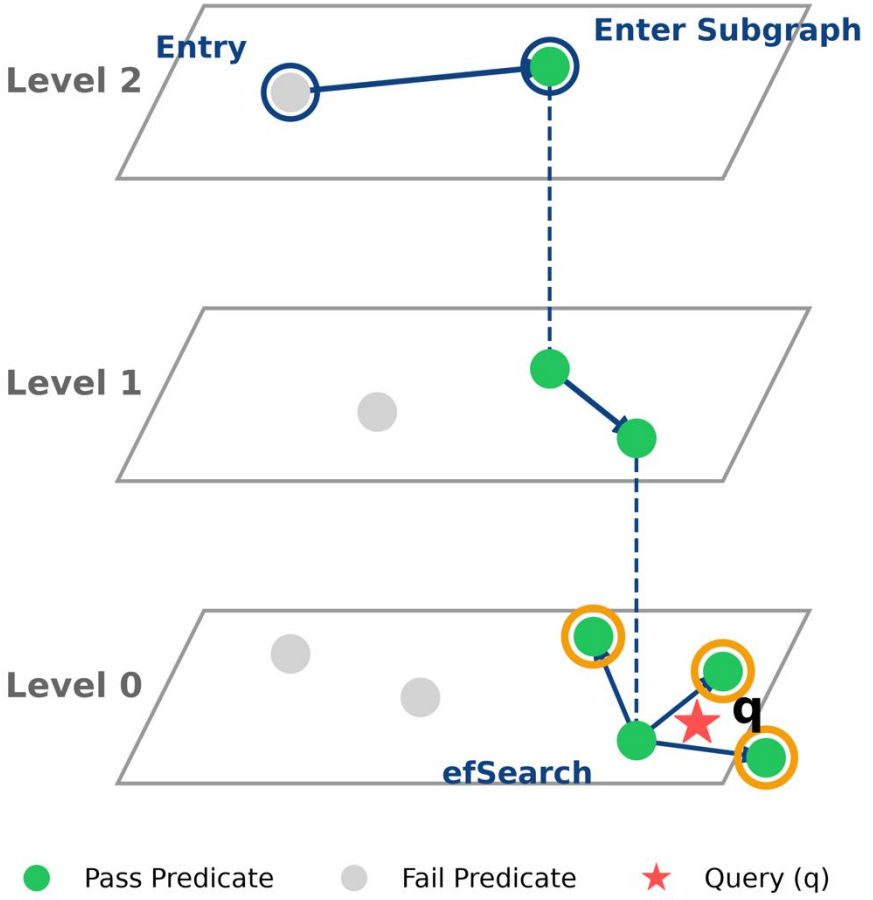
Level 0 Candidate Set Expansion



UNIVERSITY OF WATERLOO

Putting It Together: ACORN Search

Return Top-3 Nearest Neighbors



UNIVERSITY OF WATERLOO

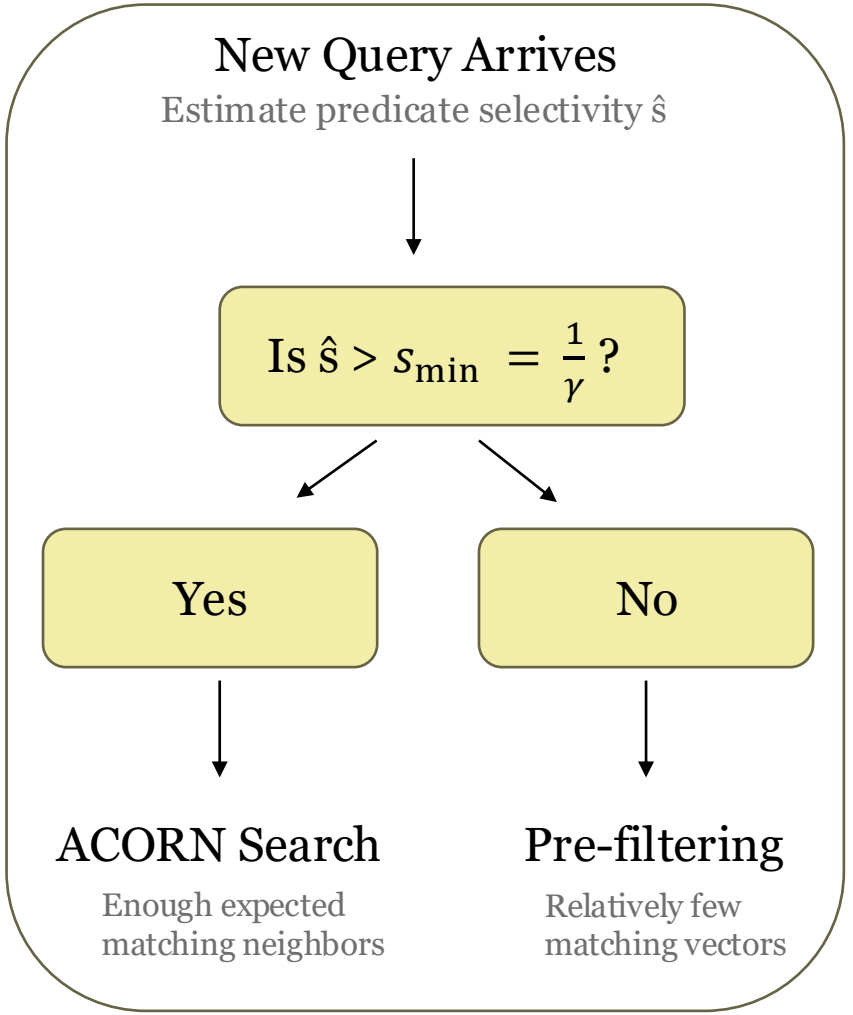
Choosing γ: Selectivity and Fallback

$$\mathbb{E}\left[\left|N_p(v)\right|\right] \approx M\gamma s \geq \mathrm{M}, \quad \text{when } \mathrm{s} \geq s_{\min}$$

To target approximately M matching neighbors:

$$\gamma \approx \frac{1}{s_{\min}}$$

- Larger γ : More potential matching neighbors, but higher construction and memory costs.
- Smaller γ : Cheaper index, potentially weaker connectivity.
- Fall back to pre-filtering for very low selectivity.



UNIVERSITY OF WATERLOO

Complexity Analysis: ACORN vs. Baselines

$\boldsymbol{d}$: Vector Dimension, $\boldsymbol{n}$: Dataset Size, $\boldsymbol{s}$: Predicate Selectivity

Method	Search Complexity
Pre-filtering	$O(sn)$
Post-filtering	Up to $O(n)$
Oracle Partition	$O(\log(sn))$
ACORN−γ	$O(\underbrace{(d+\gamma)\log(sn)}_{\text{Predicate Subgraph Search}} + \underbrace{\log(1/s)}_{\text{Entry Overhead}})$

Metric	Standard HNSW	ACORN-γ
Expected construction time	$O(n \log n)$	$O(n\gamma \log n \log\gamma) \rightarrow n \times \underbrace{O(\log n)}_{\text{Levels searched per insertion}} \times \underbrace{O(\gamma \log\gamma)}_{\text{Cost per level}}$
Memory Per Node	$O(M)$	$O(\underbrace{M_\beta + M}_{\text{Bottom level}} + \underbrace{m_L M\gamma}_{\text{Upper level}})$

UNIVERSITY OF WATERLOO

Why Does ACORN Work?

ACORN aims to emulate four HNSW properties:

- Hierarchical graph structure
- Sufficient matching neighbors
- Efficient predicate-subgraph entry
- Subgraph connectivity

Probabilistic Analysis

- Expected matching neighbors: $\mathbb{E}\left[\left|\boldsymbol{N_p(v)}\right|\right] = \boldsymbol{M\gamma s}$
- Probability of encountering a node with no matching neighbors along the search path:

$$P_{\text{zero-neighbor on path}} \leq O(\log n(1-s)^{M\gamma})$$

- The bound decreases exponentially with γ.

UNIVERSITY OF WATERLOO

Theoretical Guarantees and Limitations

What the bound tells us

- A larger γ reduces the probability of encountering nodes with no matching neighbors, under the stated assumptions.

What it does not guarantee

- Global connectivity of arbitrary predicate subgraphs.
- Exact Top-K nearest-neighbor results.

UNIVERSITY OF WATERLOO

Experimental Setup and Datasets

Low-Cardinality Predicates

Dataset	Vectors	Dimension	Predicate Type
SIFT1M	1M	128	Equality
Paper	2M	200	Equality

High-Cardinality Predicates

Dataset	Vectors	Dimension	Predicate Type
TripClick	1.06M	768	Keywords, dates
LAION-1M	1M	512	Regex, keywords
LAION-25M	24.65M	512	Regex, keywords

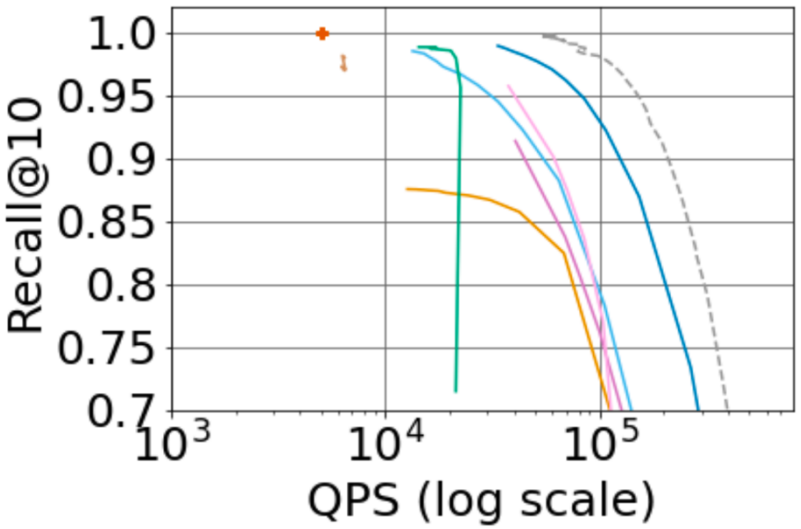
Recall@10 vs. Queries Per Second (QPS)

UNIVERSITY OF WATERLOO

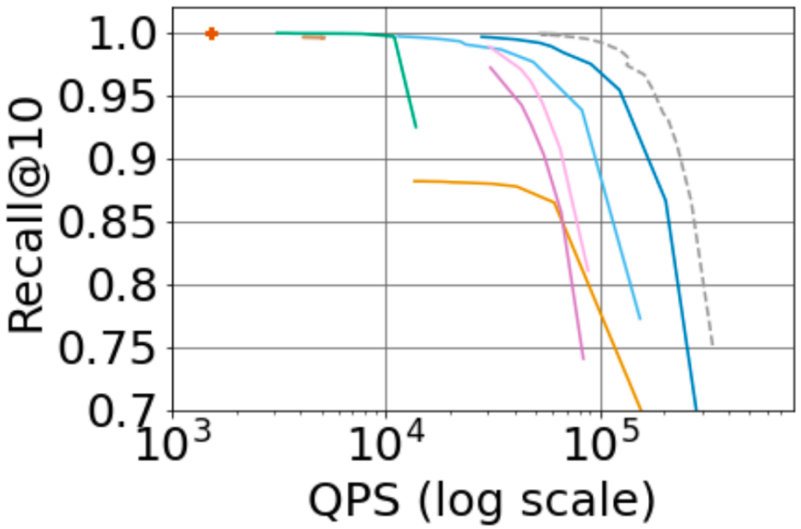
Does ACORN Approach the Oracle Partition?

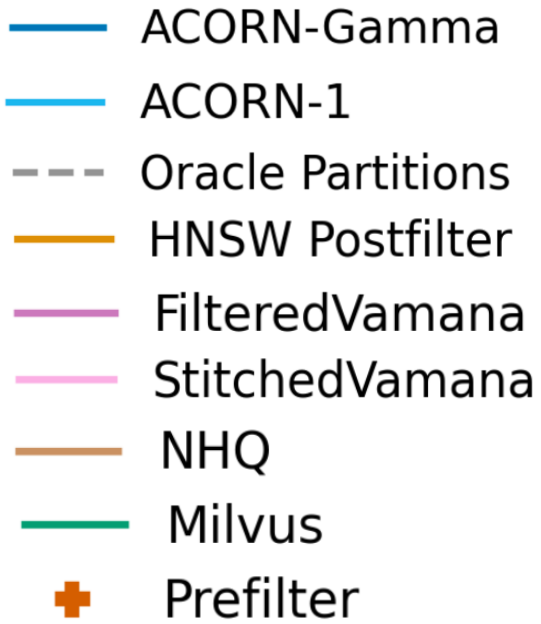
- Compare ACORN-γ, ACORN-1, and Oracle Partition.
- ACORN-γ approximates oracle-partition search efficiency.
- ACORN-γ achieves 2–10× higher QPS than competing methods on low-cardinality benchmarks at fixed recall.

SIFT1M Dataset



Paper Dataset

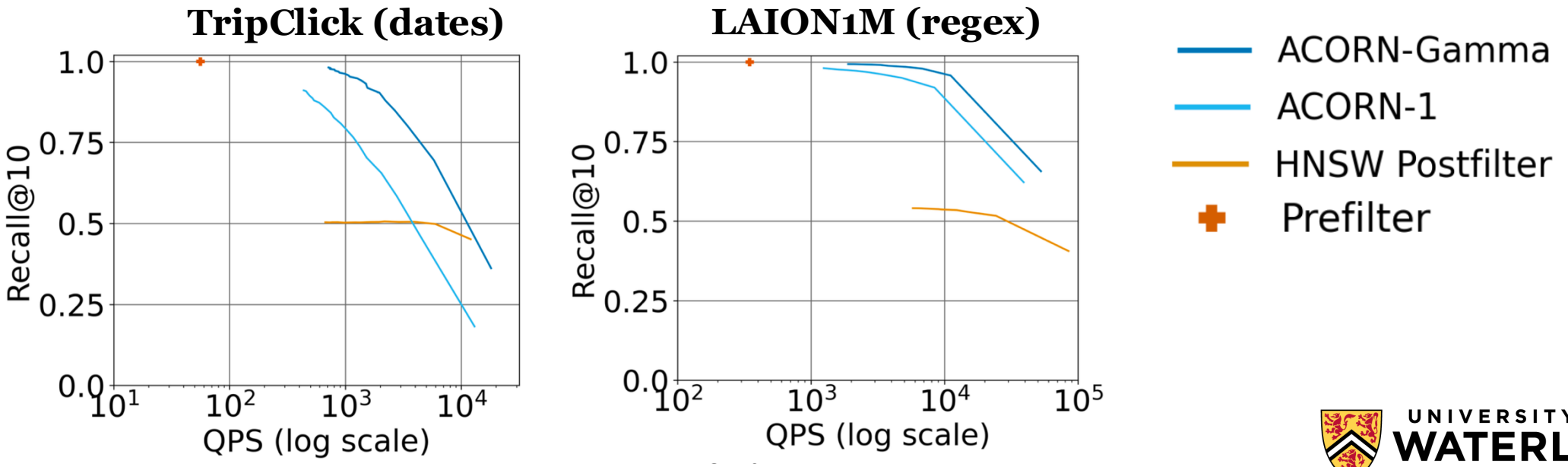




UNIVERSITY OF WATERLOO

Performance on Complex Predicates

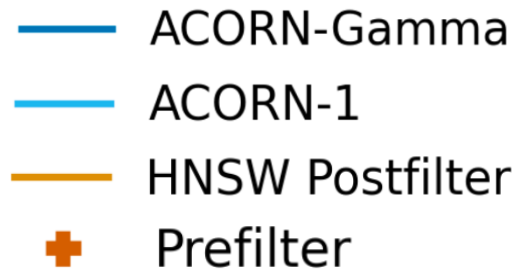
- Evaluate queries with complex, high-cardinality predicates.
- TripClick: Keyword and publication-date filters.
- LAION: Regex and keyword matching.
- ACORN achieves high recall with substantially higher throughput; the authors report 30–50× QPS improvements on these benchmarks



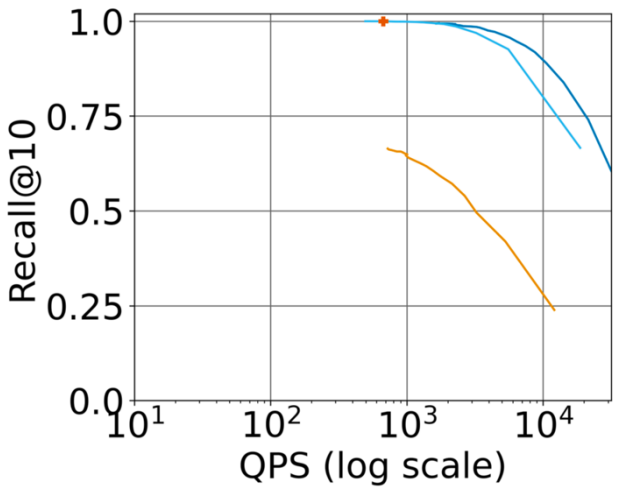
UNIVERSITY OF WATERLOO

Robustness to Predicate Selectivity

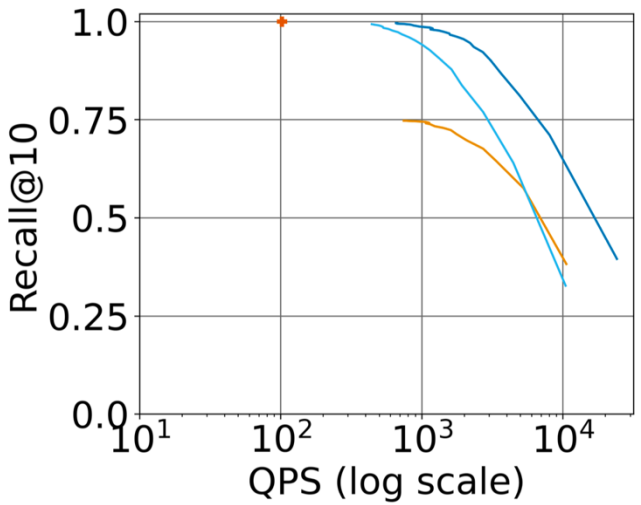
- Predicate selectivity varies from very low to relatively high.
- Pre-filtering is more competitive when few vectors match the predicate.
- Post-filtering can become expensive when few nearby nodes match.
- ACORN maintains strong search performance across selectivities.



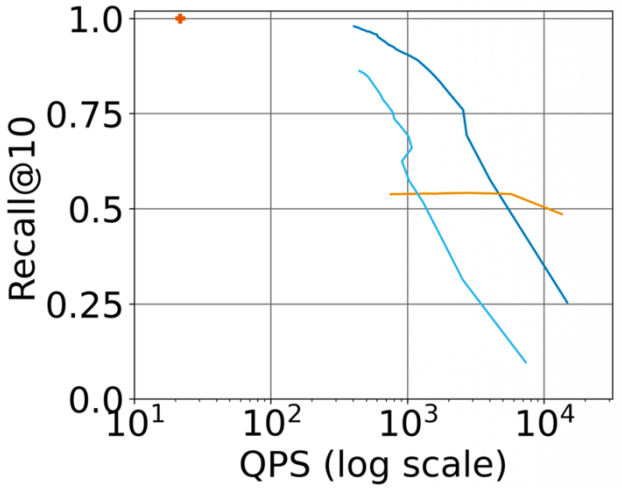
1p Sel (s=0.0127)



50p Sel (s=0.1215)



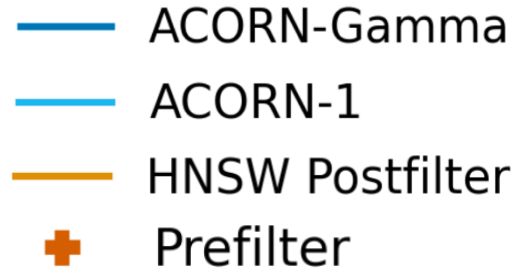
99p Sel (s=0.6164)



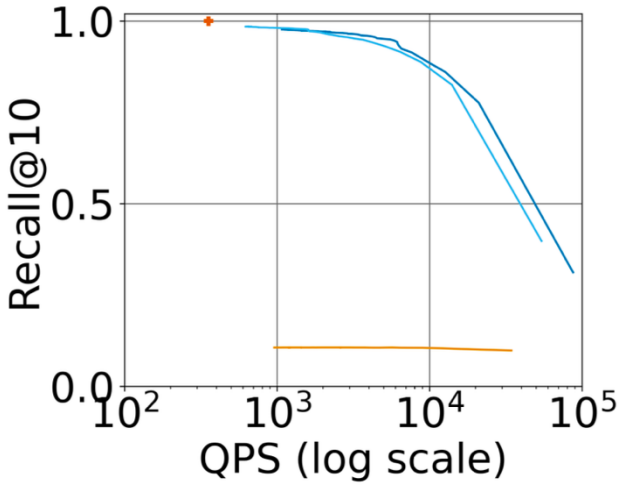
UNIVERSITY OF WATERLOO

Robustness to Query Correlation

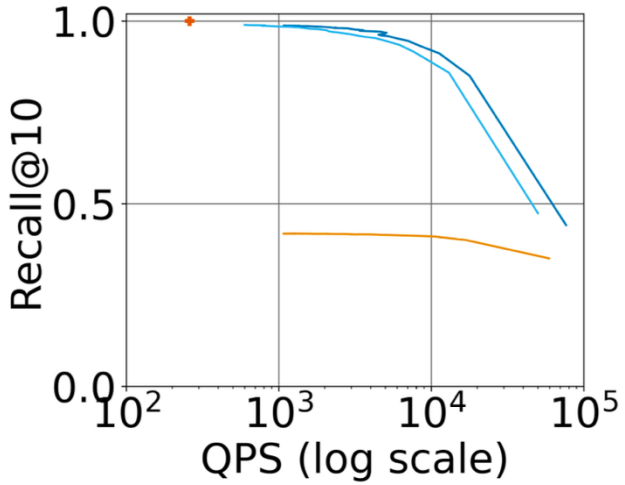
- Negative correlation: Matching vectors are far from the query.
- Post-filtering suffers from large search expansion.
- ACORN maintains high throughput across correlation patterns.
- ACORN-γ achieves 28–100× higher QPS than the best competing baseline at 0.9 recall in these workloads.



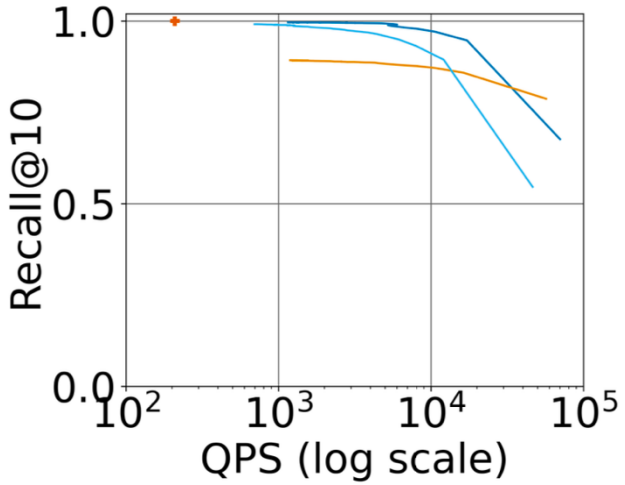
Negative Correlation



No Correlation



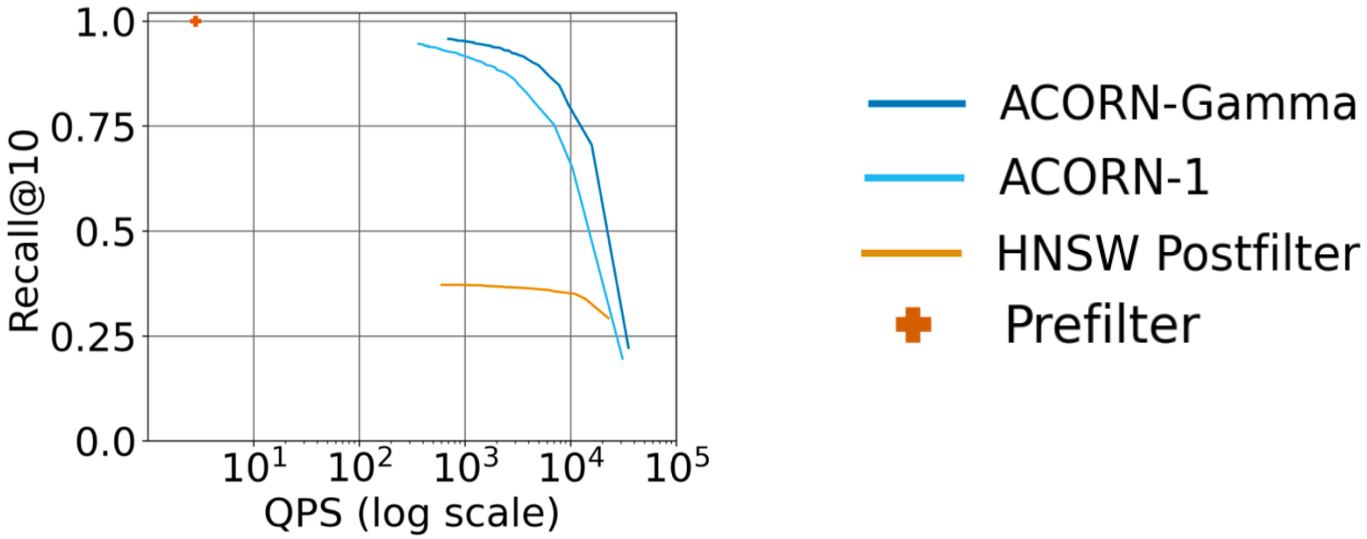
Positive Correlation



UNIVERSITY OF WATERLOO

Scalability: From 1M to 25M Vectors

- Dataset: LAION-25M.
- Approximately 24.65 million vectors.
- No-correlation query workload.
- ACORN-γ achieves over 1,000× higher QPS than the next-best baseline at 0.9 recall in this experiment.



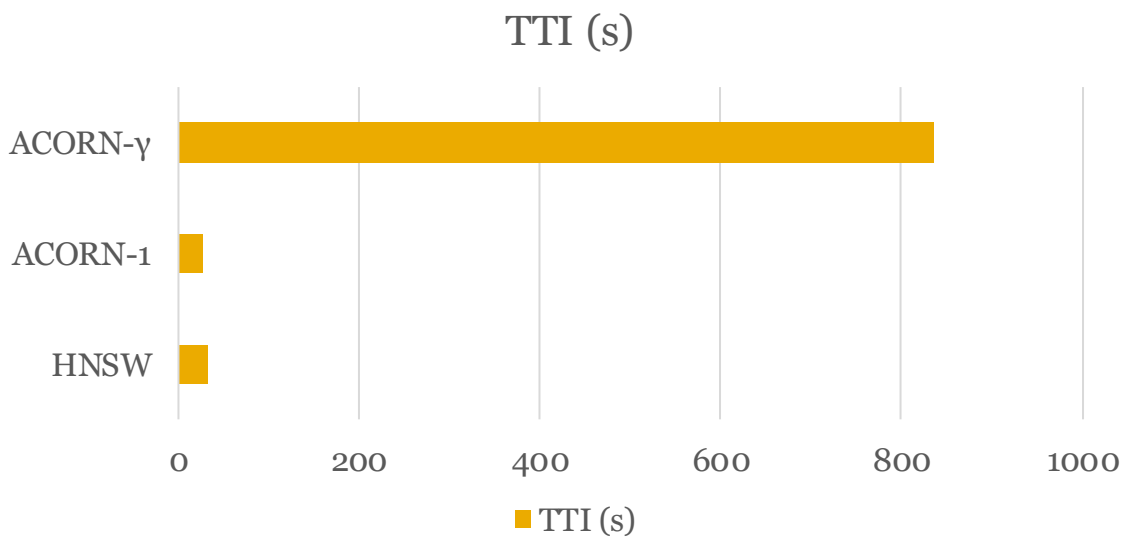
UNIVERSITY OF WATERLOO

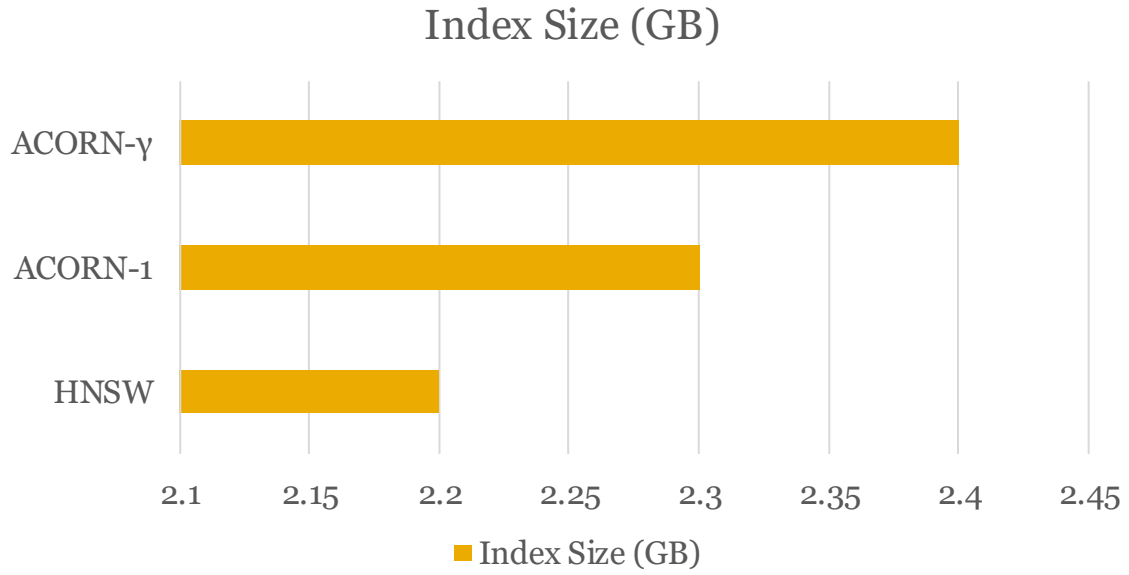
The Cost of Performance: Indexing Time and Memory

LAION-1M Example

Method	Time-to-Index	Total Size
HNSW	32.9 s	2.2 GB
ACORN-1	25.9 s	2.3 GB
ACORN-γ	835.8 s	2.4 GB

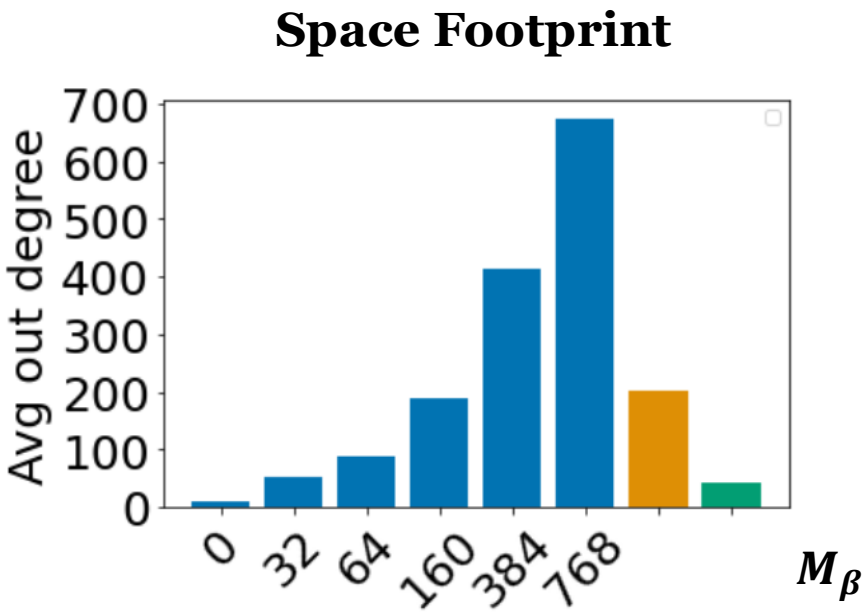
- ACORN-γ has substantially higher construction overhead.
- ACORN-1 reduces TTI while approximating ACORN-γ's search performance.
- Compression helps limit memory growth.

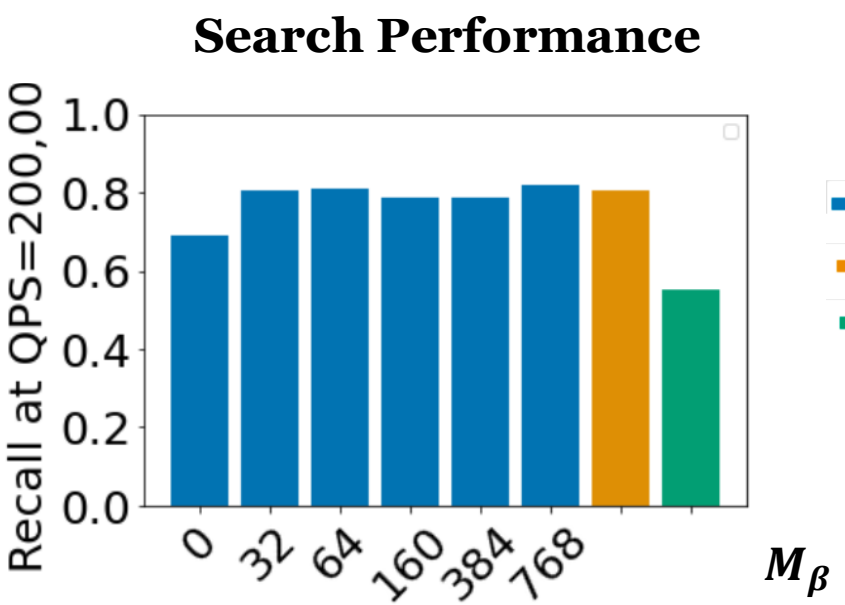


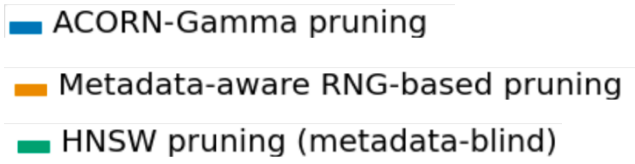


Does Predicate-Agnostic Compression Work?

- ACORN's compression reduces stored edges and construction overhead.
- Aggressive HNSW-style pruning degrades hybrid search performance.
- Predicate subgraphs empirically preserve useful graph structure.
- Graph connectivity remains an important limitation.







UNIVERSITY OF WATERLOO

VBASE — The Problem with Top-K Interfaces

- Traditional vector systems expose a Top-K search interface.
- Filtering afterward requires choosing a larger K'.
- Too small: insufficient results.
- Too large: unnecessary computation.
- VBASE avoids predicting K' in advance.

Example: Find Top-3 similar shoes under $50

Top-3 candidates

Rank	Price	Pass?
1	$89	✖
2	$69	✖
3	$70	✖

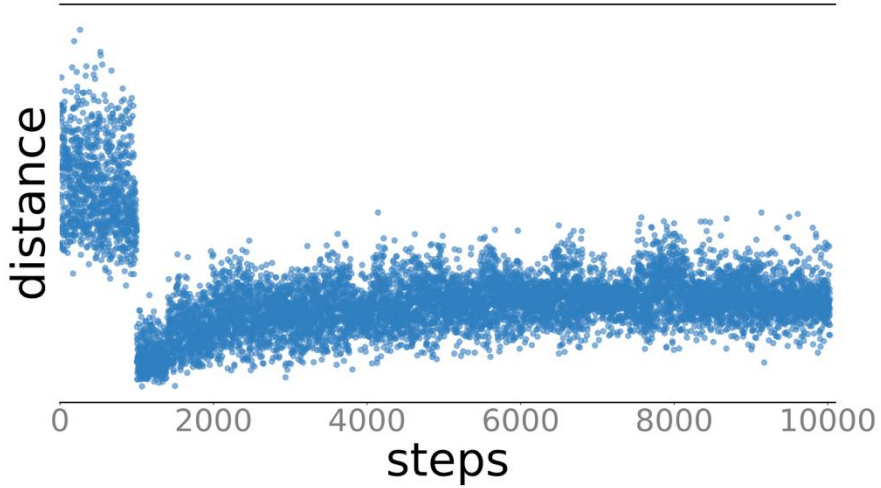
Top-100 candidates

Enough valid results

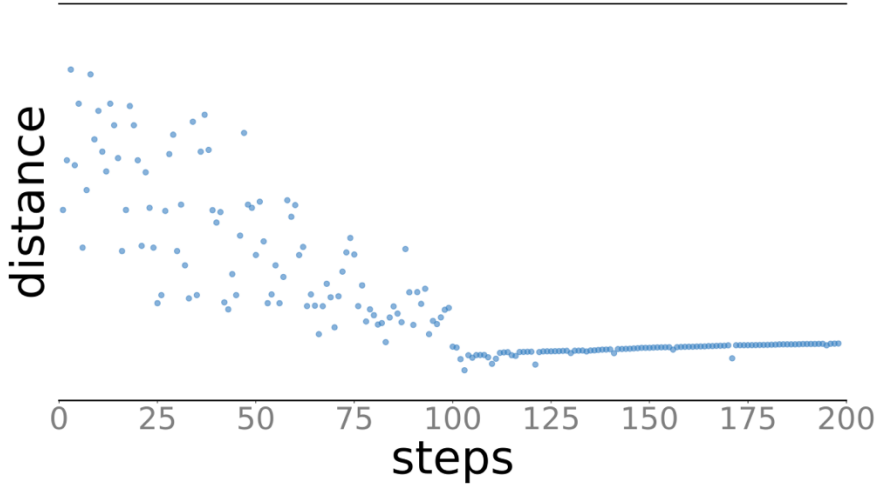
But extra candidates

UNIVERSITY OF WATERLOO

VBASE — Relaxed Monotonicity and Iterative Search



(a) FAISS IVFFlat



(b) HNSW

Two-Phase Traversal

- Phase 1 **Approach**:
  Find the region near the query.
- Phase 2 **Move Away**:
  Traversal stabilizes and gradually leaves the query's nearest-neighbor region.
- Relaxed Monotonicity: Detect when further search is unlikely to find closer vectors.

Iterative Next() Interface

Retrieve candidates incrementally
→ Apply filters
→ Check when to stop

UNIVERSITY OF WATERLOO

ACORN vs. VBASE — Two Different Approaches

Dimension	ACORN	VBASE
Main focus	Vector Index Design	Database Query Execution
Key idea	Predicate Subgraphs	Relaxed Monotonicity
Main technique	Neighbor Expansion + Compression	Iterative Next() Interface
Filtering	During Graph Search	During Query Execution
Main benefit	Efficient Filtered ANN	Flexible Vector + SQL Queries
Limitation / Cost	Higher Indexing Costs	Relies on Underlying ANN Index

ACORN optimizes the index; VBASE optimizes query execution.

UNIVERSITY OF WATERLOO

Key Takeaways

- **Predicate-Subgraph Traversal**

  ACORN focuses graph search on matching vectors, reducing unnecessary exploration.

- **Predicate-Agnostic Indexing**

  ACORN-γ expands neighbor lists and uses compression without knowing future query predicates.

- **Search–Construction Trade-off**

  ACORN-γ prioritizes search efficiency, while ACORN-1 reduces indexing overhead.

- **Two Approaches to Hybrid Vector Search**

  ACORN improves vector indexing, while VBASE improves database query execution.

UNIVERSITY OF WATERLOO