An In-Depth Study of

Filter-Agnostic Vector Search on a PostgreSQL Database System

Experiments & Analysis · SIGMOD 2026

Duo Lu[1]* · Helena Caminal[2] · Manos Chatzakis[3]* · Yannis Papakonstantinou[2] · Yannis Chronis[2 4] · Vaibhav Jain[2] · Fatma Özcan[2]

[1]Brown University [2]Google [3]Université Paris Cité, LIPADE [4]ETH Zürich *work done while at Google

Motivations

Filtered Vector Search now runs inside the DBMS — but it has only been studied in Libraries

Research & Benchmark Gap

Research has not addressed FVS inside a real DBMS. Its algorithms — *ACORN, NaviX, Sweeping, ScaNN* — were all evaluated in standalone main-memory libraries, and FVS still **lacks a comprehensive, widely-accepted benchmark**

Performance Gap

Big gap between db and memory implementations
Variable gap: **Selectivities** and **algorithms** affect it

What is Filtered Vector Search?

A single query pairs processing over **structured attributes** with a **semantic vector-similarity search**. The DBMS must plan and run both together

```
SELECT * FROM products
WHERE size = 'XL'
  AND id IN (SELECT productid
        FROM inventory WHERE near_home)
ORDER BY descr_vec <=> embed('blue shirts')
LIMIT 10;
```

filter ⋀ join + vector top-k ⟶ merged result

> FVS Used across pgvector, Oracle, Mongo, Elastic, Cosmos DB — yet the FVS algorithms were all evaluated in main-memory libraries, not a DBMS

The FVS Index Landscape

Four *filter-agnostic* strategies

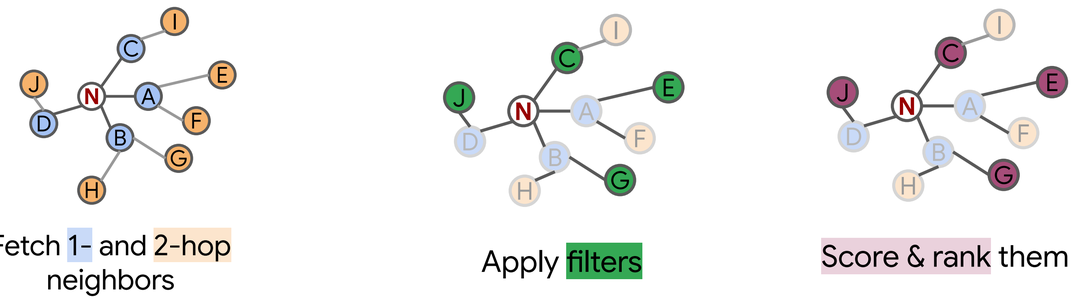
	FILTER-FIRST	TRAVERSAL-FIRST
GRAPH HNSW family	**NaviX / ACORN** predicate-subgraph traversal; 2-hop neighbor expansion	**Sweeping / Iterative Scan** unmodified HNSW; filter candidates after scoring
TREE clustering	**ScaNN — inline filtering** sequential leaf scan; filtering mixed with distance comps	**ScaNN — post filtering** unmodified ScaNN, followed by filtering

> Filter-agnostic = the index doesn't know the filter at build time. No rebuilds when query patterns change — attractive for production DBMS!

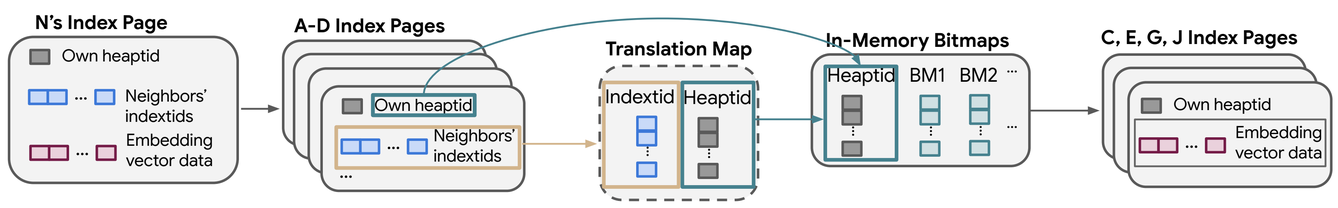
1 The "System Tax" — Algorithms have different levels of "tax" when built in DBs

Example: ACORN on HNSW

Algorithmic View



System View



(Optimization)

Translation Map: custom in-memory hash map

55–91% of CPU cycles are **system overhead** — page access, data retrieval etc...

> **Filter-first ≠ free. Architectural overhead dominates algorithm efficiency** — so library benchmarks that count only distance comps cannot predict DBMS behavior

2 Quantization only pays for sequential access

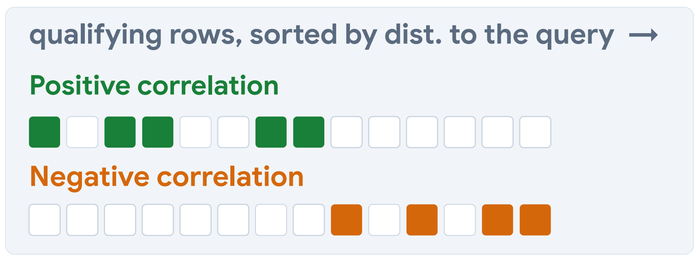
Graph · HNSW	Tree · ScaNN
Random page access	**Sequential SIMD scan**
~1.03×	**up to 50×**
QPS gain from PQ/SQ	**latency cut at 1% selectivity**
Each neighbor visit still triggers a distinct page access — **quantization can't help**	PCA + SQ8 unlock **SIMD** across contiguous leaves — quantization works when **access is sequential**

> The system aspect decides: quantization speeds **ScaNN's sequential** scans, but not **HNSW's random page access**

3 We need FVS benchmarks

A workload generator, not synthetic columns

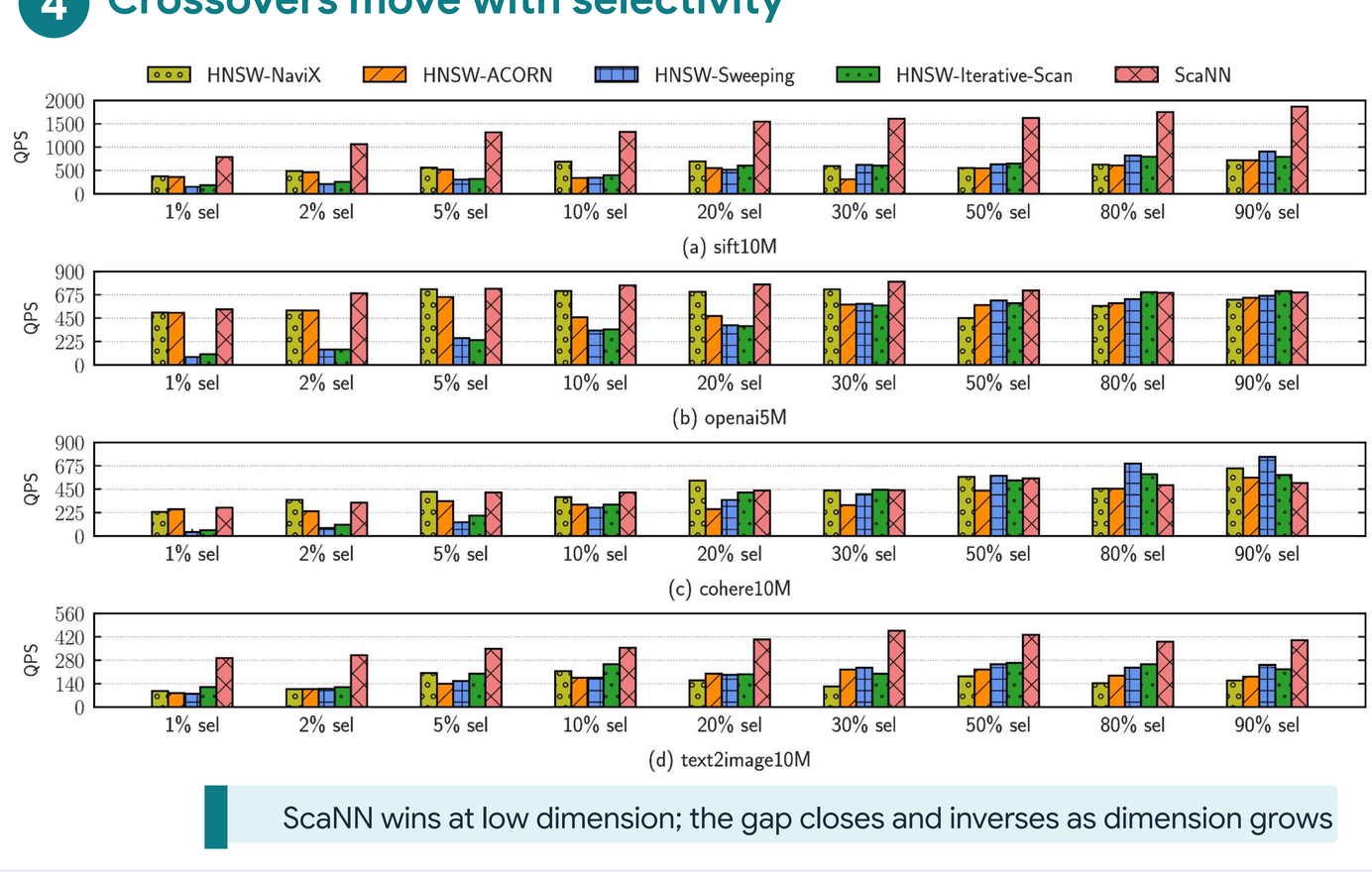
Each filter = a **bitmap of qualifying rows**.
Stress-test with adjustable ***selectivity*** and tunable ***correlation*** to the vector neighborhood



Evaluated on:

- SIFM10M, OpenAI5M, Cohere10M, Text2image10M
- commercial PG-compatible DBMS + pgvector & ScaNN
- 240 GB RAM · 16 clients · 95% Recall@10

4 Crossovers move with selectivity



> ScaNN wins at low dimension; the gap closes and inverses as dimension grows

Future Work

1. ***System-aware FVS algorithms***
2. ***New storage engines for FVS***
3. ***Rich benchmarks / workload generation***

Beyond bitmap filters: a cost-of-filtering knob for expensive predicates & compound conditions

Paper: doi.org/10.1145/3802011