An In-Depth Study of

Filter-Agnostic Vector Search on a PostgreSQL Database System

Experiments & Analysis · SIGMOD 2026

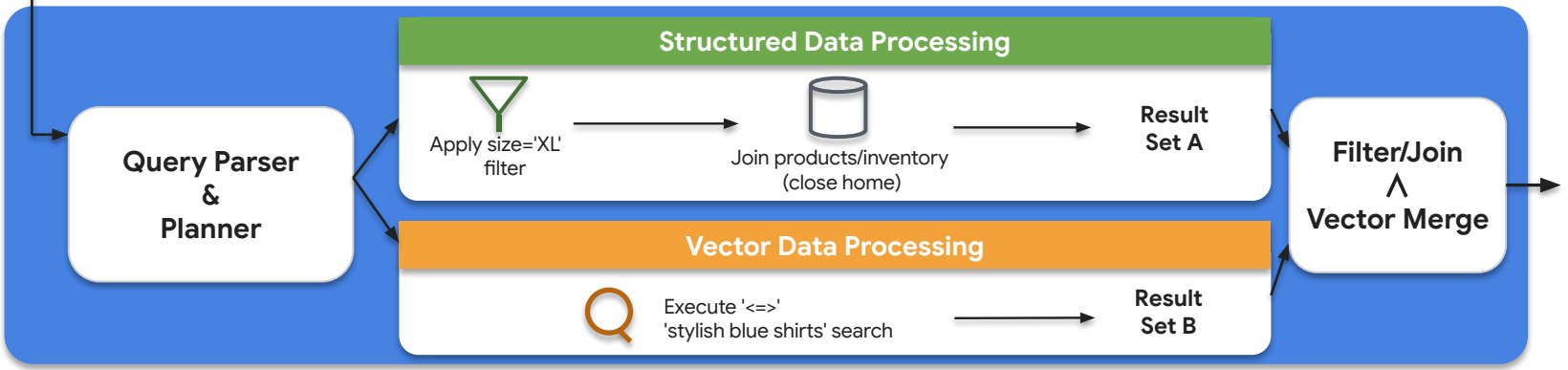
Duo Lu[1*] · Helena Caminal[2] · Manos Chatzakis[3*] · **Yannis Papakonstantinou[2]** · Yannis Chronis[2,4] · Vaibhav Jain[2] · Fatma Özcan[2]

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

SQL-based Vector Search

INPUT

```
SELECT *
FROM products
WHERE size='XL'
    AND id IN (SELECT productid FROM inventory WHERE close to my home)
ORDER BY description_vector <=> embedding('stylish blue shirts')
LIMIT 10;
```



OUTPUT

Result Set A (filter pass)

ID	Description	Size
88	Indigo Cotton Button-Down	XL
101	Slim-Fit Blue Shirt	XL
142	Red Cotton T-Shirt	XL
203	Navy Linen Shirt	XL

∩

Result Set B (vector top-k)

ID	Description	Score
101	Slim-Fit Blue Shirt	0.99
56	Casual Blue Polo	0.98
88	Indigo Cotton Button-Down	0.97
112	Royal Blue Tee	0.96

→

Output (A ∩ B)

ID	Description	Score	Size
101	Slim-Fit Blue Shirt	0.99	XL
88	Indigo Cotton Button-Down	0.97	XL

Research has not addressed Filtered Vector Search (FVS) in DBs
FVS also lacks comprehensive, widely-accepted benchmarks

- **filter(+join)+vector search in SQL databases is common and useful**
- PostgreSQL/pgvector, Oracle, Mongo, Elastic, Cosmos DB

filtered vector search algorithms live in main memory libraries

ACORN, *NaviX*, *Sweeping*, *ScaNN* — all evaluated in **standalone main memory libraries**. Do those conclusions transfer to a real DBMS?

> At Target, we used AlloyDB to improve our online search experience. We used the ability to combine our structured and unstructured data to enhance the accuracy of natural language search queries by 20%!"
>
> Visagan Subburayalu, VP of Infrastructure & Cybersecurity, Target

Big gap between db and memory implementations

Variable gap: Selectivities and algorithms affect it

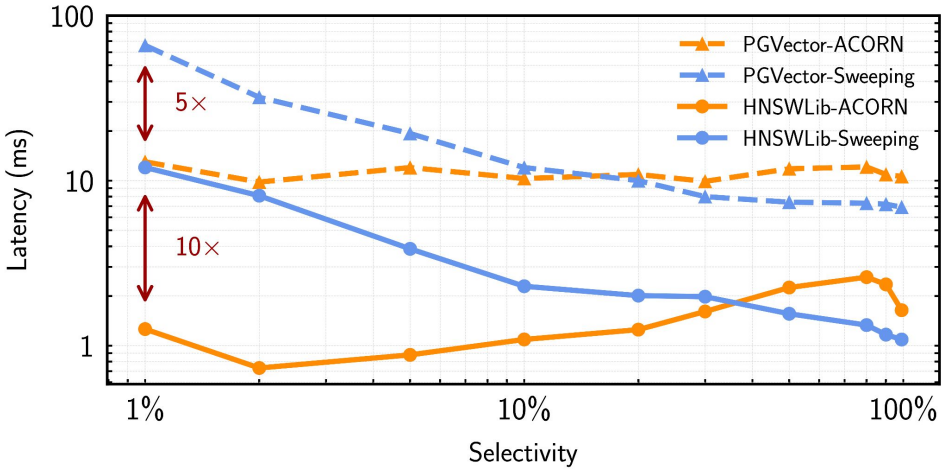


Fig. 1. Average latency for 100 filtered vector search queries on OpenAI-1M: HNSWLib (solid) vs. PGVector (dashed). System latency can be up to 10× higher, and the gap varies with selectivity, motivating an in-depth comparison of FVS algorithms in a real DBMS.

In a real DBMS

10×

latency gap vs. a standalone library

And the gap isn't constant

The crossover between algorithms moves with selectivity — pick from a library benchmark, pick wrong

Focus on Filter-Agnostic Methods

What is a Filter-Agnostic Index?

The index structure is built without any knowledge of the structured data filters that will be applied at query time.

Why is this attractive for DBMS?

- **Universal Compatibility:** Works with any combination of filter predicates at runtime.
- **Reduced Maintenance:** No need to rebuild indices when query patterns change.
- **Production Ready:** Simplifies deployment in dynamic database environments where filters vary per user.

Key for production DBs: The index doesn't know the filter at build time

Landscape of Filtered Vector Search Indexes

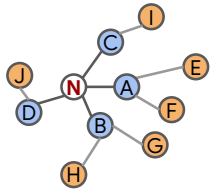
	FILTER-FIRST	TRAVERSAL-FIRST
GRAPH *HNSW family*	**NaviX / ACORN** predicate-subgraph traversal; 2-hop expansion	**Sweeping / Iterative Scan** unmodified HNSW; filter after scoring
TREE *clustering*	**ScaNN - inline filtering** sequential leaf scan; filtering mixed with distance computations	**ScaNN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

All three are filter-agnostic — the index doesn't know the filter at build time. This is attractive for production DBs!

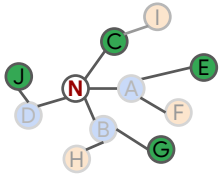
Lesson #1: The “System Tax” (on Dereferencing)

Example: ACORN on HNSW

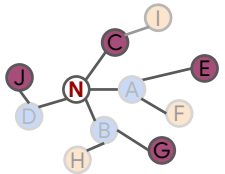
Algorithmic View



Fetch 1- and 2-hop neighbors

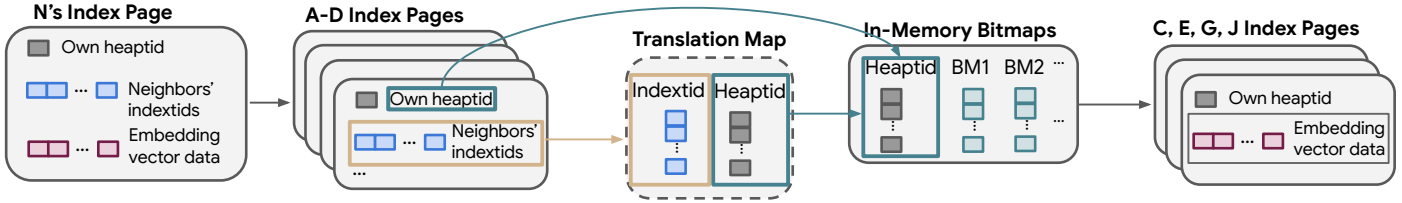


Apply filters



Score & rank them

System View



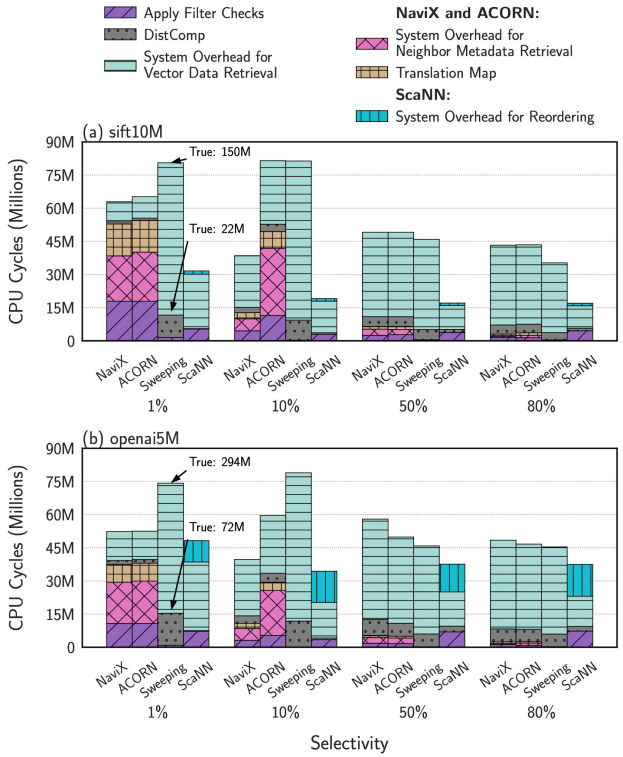
(Optimization)

Translation Map: custom in-memory hash map

~~**Library: Filters, distcomp -> Perf**~~

System: Filters, distcomp, system, index -> Perf

Most cycles are system overhead



CPU cycle breakdown, sift10M & openai5M

55 – 91%

of CPU cycles are system overhead, i.e., page access, TID indirection, tuple materialization, locking

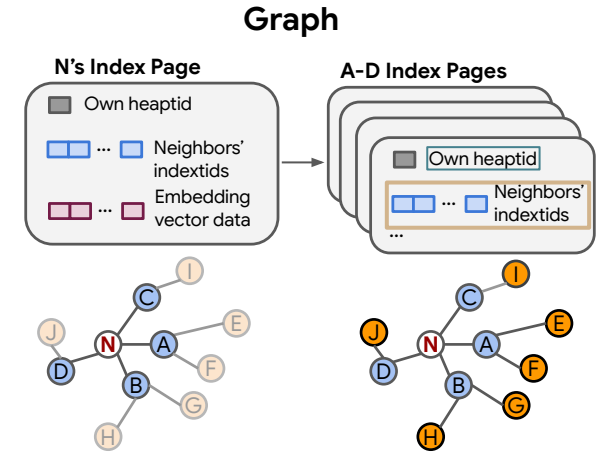
Filter-first ≠ free

Architectural overheads impact your algorithm efficiency!

This is why benchmarks focus on just libraries are not sufficient: *e.g., distance computations (metrics that the library try to optimize) are not sufficient — page access & data retrieval dominate inside a DBMS*

Lesson #2: Quantizations do not matter when navigation costs

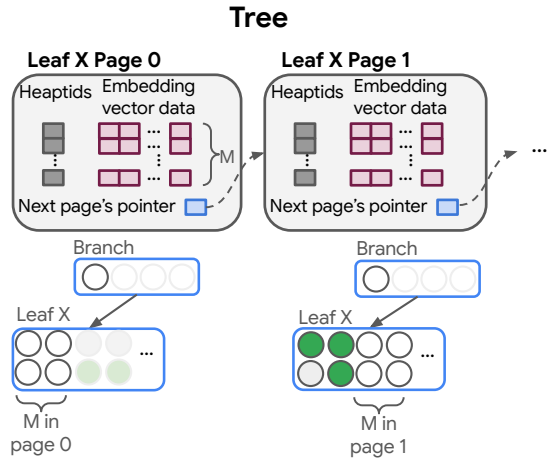
The system aspect: it speeds ScaNN's sequential SIMD scans, but not HNSW's random page access



Dataset	Method	QPS	Index Size	Build Time
sift10M	Halfvec	0.97×	1.34×	1.01×
	BQ+rerank	–*	2.00×	1.02×
openai5M	Halfvec	1.03×	2.00×	1.11×
	BQ+rerank	0.75×	11.63×	2.86×
cohere10M	Halfvec	1.04×	2.00×	1.24×
	BQ+rerank	0.96×	11.83×	2.33×
text2image10M	Halfvec	0.99×	1.53×	1.14×
	BQ+rerank	–*	2.55×	1.17×

*Unable to reach 95% recall within reasonable search parameters.

Table 4. QPS speedup and reduction in index size and build time of 1,000 unfiltered vector search queries in PGVector-HNSW with 16 threads (Microbenchmark 95% Recall@10).



Dataset	Method	1% sel.	5% sel.	20% sel.	50% sel.	80% sel.
sift10M	Quant.	1.40×	1.42×	1.56×	1.49×	1.58×
openai5M	PCA	29.39×	5.79×	3.92×	3.22×	3.62×
	PCA+Quant.	50.21×	8.61×	6.06×	5.02×	5.17×
cohere10M	PCA	5.24×	4.47×	3.48×	3.76×	3.83×
	PCA+Quant.	9.60×	7.27×	6.55×	6.48×	5.52×
text2image10M	Quant.	2.17×	1.51×	1.45×	1.87×	1.79×

*sift10M and text2image10M are low-dimensional datasets. PCA is not applicable.
*PCA vector dimension reduction (openai5M: 1536→193, cohere10M: 768→157).

Table 5. Latency speedup of different settings w.r.t. non-quantized non-PCAed ScaNN on 95% recall@10 with 16 threads (no correlation). Columns show increasing selectivity. Quantization uses SQ8.

Lesson #3: We need Filter Vector Search Benchmarks

Our step: Workload Generator & Datasets

*Tune the filter selectivity and **correlation** knobs freely*

Our workload generator:

1. No synthetic attribute columns needed!
2. Systematic stress testing:
   a) Adjustable *selectivity*
   b) Tunable *correlation*

High-dimensionality + Large volume

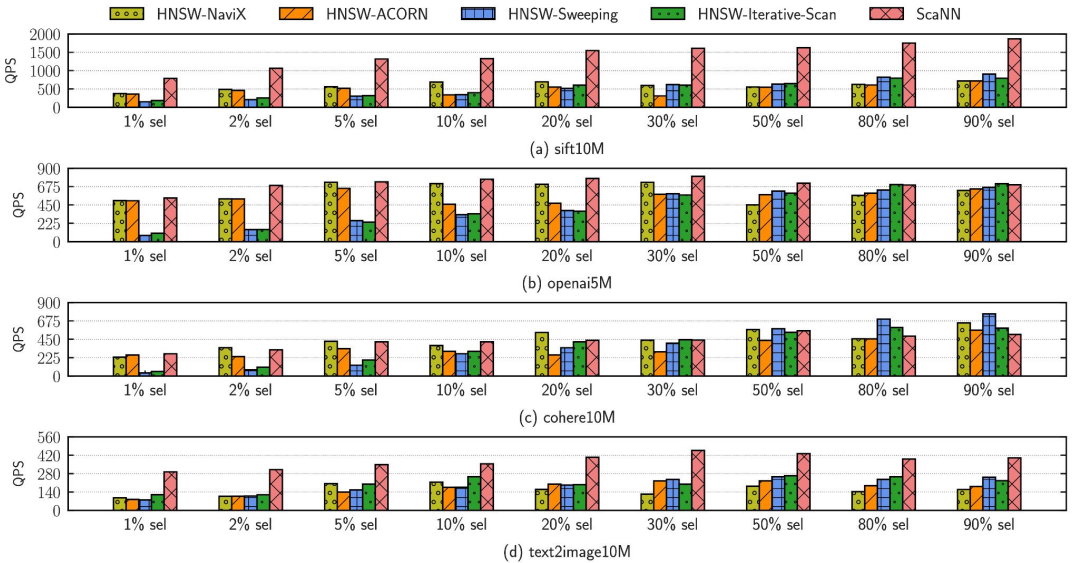
Most difficult: OOD queries

Query hardness

Dataset	# Vec	Dims	Metric	Source	Dist.-	Rel. Cost	LID/LRC
sift10M [26]	10M	128	L2	Image Descriptors	0.8x		19.11/0.82
openai5M [6]	5M	1536	IP	Text Embeddings	6x		33.34/0.97
cohere10M [1]	10M	768	L2	Text Embeddings	3.3x		38.63/0.97
text2image10M [46]	10M	200	L2	Multimodal Emb.	1.3x		52.47/0.98

Lesson # 4

Crossovers move with selectivity



QPS @ 95% Recall@10 · no correlation · 4 datasets

1

Filter-first wins low selectivity

NaviX / ACORN tunnel through the predicate subgraph — but lose to Sweeping past *~10-30%*

2

ScaNN wins low-dim, gap closes high-dim

Sequential leaf scans beat random graph walks on sift / text2image; openai / cohere converge

Future Work on Vector Search in DBs:
R&D call from Vector Search Panel at VLDB 2024

-> The database is the system of record:
transactional semantics & strong consistency

-> Has to work for general OLTP workloads: reads + writes

-> Has to work in disk and exploit memory for performance

-> Achieve sql vector index performance that effectively matches memory-based native index performance when given enough memory

AlloyDB employed its columnar engine to host both the index and the filter columns; cut page-access overhead while keeping ACID and out-of-core support

Future Work

1. **System-aware FVS algorithms**

2. **New storage engines for FVS**

3. **Rich benchmarks / workload generation**

   Beyond bitmap filters: an adjustable cost-of-filtering knob to model expensive predicates and compound conditions

Thank you

Filter-Agnostic Vector Search on a PostgreSQL Database System

Questions?

Yannis Papakonstantinou · yannispap@google.com

Paper: doi.org/10.1145/3802011