MorletDB: Recordings, Not Rows — Lag-Tolerant Waveform Retrieval in PostgreSQL

Version 0.31

Seung-Zin Nam
Germany

Semir Maslo
Sulzer-BDiM
Germany

Abstract

Machines, patients and structures are increasingly monitored by sensors that record long waveforms, and engineers and AI agents ask of these archives: where did this machine, or another one, behave like this before, and what exactly did it record then? Answering needs the original samples, a similarity that tolerates small time shifts, and the filters, access control and transactions of a database; today these come from separate systems. MorletDB is a PostgreSQL extension that stores recordings bit for bit and searches them under a bounded time shift. Ordinary vector search asks the wrong question: on held-out data, zero-lag shortlists recovered only 61–66% of the shift-tolerant top ten, exact or approximate, so the metric, not the approximation, loses the matches. MorletDB instead turns each shift into one inner-product query against a single, unmodified vector index (thirteen probes, no training, no extra copies) and re-ranks the candidates with the exact scorer. On held-out seismic data, probes recovered 97.7% of the exact top ten at a p95 latency of 57 ms against 740 ms for exact search, and in a later study stayed 1.6–2.3× faster than a screened exact search with identical results on up to one million windows. Matches were also better by a catalogue proxy (exact search, held-out data): 0.139 of bounded-lag top-ten positions held a catalogued earthquake's arrival, against 0.060 for zero lag. Every hit is a sample interval that reads back the original samples, under the same filters and row security as exact search; appends are atomic, and twenty writers killed mid-append left no partial rows. The probe structures cost 0.97–1.34× the raw data. On public CNC milling data, bounded-lag similarity separated bad runs better than zero-lag similarity, but run length and signal level did as well or better; MorletDB therefore serves shape, level and context together rather than treating similarity as a diagnosis. Code, frozen protocols and raw results: https://github.com/merkhet-io/morletdb.

1 Introduction

Sensors now record machines, patients and structures continuously: vibration and motor current on machine tools in smart manufacturing, physiological signals in health monitoring, ground motion in seismic networks. These archives of long, uniformly sampled recordings are worth keeping because they can be compared. When a machine sounds wrong, the first questions are where it, or another machine, behaved like this before and what the signal looked like then; increasingly an AI agent asks them on an engineer's behalf. Three things are needed for a useful answer: the original samples, kept bit for bit as evidence; a similarity that tolerates small time shifts; and the guarantees of a database (filters, access control, transactions) that make a hit trustworthy. General-purpose stacks spread them over a time-series database, a vector index and scripts, and each hand-off risks losing one of the three. MorletDB keeps all three in PostgreSQL. Table 1 summarises what it delivers and what it costs; condition monitoring of machines motivated the work.

Table 1: MorletDB at a glance: each advantage with its measured result, the comparison it was measured against, and where the evidence is. All figures are measured on the stated data only.

Advantage	Measured result (against)	Sec.
Fast shift-tolerant search	97.7% of the exact top ten at p95 57 ms (exact search: 740 ms); held-out seismic data	7
Faster than the best exact path	1.6–2.3× at p95, recall 0.99–0.998, 0.23–1.03 M windows (screened exact search, identical results)	7
Why a plain vector index is not enough	held-out zero-lag shortlists recover 61–66% (probes: 97.7%), exact or approximate alike	6
Better matches	more top-ten windows hold a catalogued earthquake's arrival than with zero lag, +0.08 on dev and held-out data (exact search; a proxy)	12
Lossless storage	bit-exact float64; CNC counts at 9.93 bits per sample (float64: 64); smallest of the codecs tested on most waveform inputs (Fig. 2)	4
One contract on both paths	same filters, row security and sample intervals for exact and probe search; windows searchable at commit	4
Small machines	full workflow in 2 CPUs and 512 MB; runs on a Raspberry Pi 3B+	4
Industrial evidence	bad CNC runs: AUROC 0.925 (zero lag 0.875), but run length 0.982 (Fig. 7)	10
Costs	probe structures 0.97–1.34× raw data; exact search for small or selective galleries; write-ahead log 2.9–18× database growth during ingestion	8, 9

A common investigation starts from one interval and asks where similar behaviour occurred, then inspects the original samples and acquisition context of each candidate. Two practical requirements follow. Similarity must tolerate a small time shift, because a window grid rarely aligns two occurrences of the same event; and a result is useful only if it identifies an interval of a recording that can be read back exactly.

Industrial requirements and lessons. The collaboration with Sulzer-BDiM motivates four requirements: retain the original evidence after a search, restrict comparisons to compatible acquisition context, keep newly built search windows transactionally visible, and provide interactive investigation on constrained machines. These are design requirements, not claims of a customer deployment. Section 10 evaluates the system on public machining data and exposes a consequential limitation: normalising away level and duration can remove the strongest condition information. The engineering contribution is to serve these quantities beside shape,

with a common interval and provenance, and to make the storage, latency and recall costs explicit.

A bounded-lag cross-correlation captures the first requirement: two windows are similar if some shift of at most L samples correlates them strongly (Section 2). Exhaustive evaluation is linear in the archive. The obvious acceleration is a vector index, which PostgreSQL users already have through pgvector [1]; but such indexes rank by a fixed metric between unshifted vectors. This paper asks:

> *How can bounded-lag waveform search be accelerated with standard vector indexes while preserving the link to the recording, the semantics of filters and transactional updates, and which costs limit its use?*

The answer has three parts. Zero-lag vector search asks the index the wrong question: it ranks by a different quantity and loses a quarter of the scorer's top ten, and an exact zero-lag shortlist loses the same windows as an approximate one (Section 6). The right question is a set of shifted inner-product queries: each lag term of the scorer is an inner product with a shifted query, so a few shifted probes against one index generate candidates for the scorer actually used (Section 3). And PostgreSQL can answer it without giving up what makes a hit useful: candidates are re-ranked with the scorer and returned as sample intervals of losslessly stored recordings, under the same filters and access rules as exact search (Section 4).

Contributions.

(1) **A retrieval diagnosis.** Zero-lag shortlists fail for a bounded-lag scorer because of their metric, not because of approximate search: exact and HNSW shortlists lose the same matches (Section 6).
(2) **A simple method.** One unmodified inner-product HNSW index serves a bounded-lag scorer: query-side, zero-padded shifted probes on a sparse lag grid, the union of their candidates, and re-ranking with the unchanged scorer. It needs no trained embedding, no per-shift copies of the data and no lower bounds (Section 3).
(3) **One contract on both paths.** Exact and probe search share eligibility, metadata filters, row-level security, source exclusion and overlap suppression. Every hit is an interval that reads back the original bits, and windows become searchable at the commit that builds them. A router sends selective filters to exact search (Section 4).
(4) **Evidence and industrial lessons.** Frozen protocols, an ablation separating candidate errors, measured storage and maintenance costs, and two public machining studies that identify where shape retrieval helps and where level, duration and history drift limit it (Sections 6–10).

The main retrieval ablation is seismic; Section 10 adds machining experiments and Section 12 states the limits.

Code, frozen protocols and raw results are public [5]; a supplement reports storage, recovery and further studies.

2 Problem and search contract

A *recording* is a sequence $x_0, \dots, x_{N-1}$ of float64 samples with a sample rate, a channel, a unit and JSON acquisition metadata. An *interval* $[a, b)$ denotes $x_a, \dots, x_{b-1}$; sample offsets, not wall-clock times, are authoritative. A *window* is an interval of fixed length w on a stride grid. A search representation $\phi(x_{a:b}) \in \mathbb{R}^n$ is derived from a window by a versioned extractor.

We study envelopes: for $w = 4096$, z-score the window, take the magnitude of the analytic signal, smooth with a 50-sample moving mean, keep every tenth value and z-score again ($n = 410$). Stored envelopes are float16. For a query envelope q computed at full precision and a stored envelope g, the scorer is

$$D(q, g) \;=\; 1 - \max_{|\ell| \le L} \frac{1}{n} \sum_t q_{t+\ell}\, g_t, \qquad L = \lfloor 0.075\, n \rfloor = 30, \quad (1)$$

where terms with $t + \ell$ outside $[0, n)$ are zero. This is a *biased* cross-correlation: no per-lag renormalization of the overlapping part, no circular shift, and one z-score per window rather than per overlapping subsequence. At 100 Hz, $L = 30$ envelope values correspond to ±3 s.

Contract. A query names a source interval (or supplies samples), a metadata filter and k. The *eligible* windows are those whose recordings have the query's sample rate, channel and unit, whose window size and extractor match, whose metadata contain the filter, and which the caller may see under row-level security. The *exact result* is the first k eligible windows in the order $(D, \text{recording}, \text{start})$, after excluding windows overlapping the source interval and greedily skipping windows that overlap a better window of the same recording. Each result row is a recording identifier, an interval and its distance; the interval reads back the original samples bit-exactly from the raw store.

Three kinds of exactness. We keep apart (i) *exact distances*: every returned distance is D computed on the stored envelope; (ii) *exact candidate selection*: the returned set equals the exact result; and (iii) *relevance*: the returned intervals help answer the analyst's question. Probe search guarantees (i) but not (ii); we measure (ii) as recall@10 against the exhaustive search; neither implies (iii).

3 Shifted inner-product probes

For a lag ℓ, define the zero-padded shifted query $s_\ell[t] = q_{t+\ell}$ if $0 \le t + \ell < n$ and 0 otherwise. Then $\sum_t q_{t+\ell} g_t = \langle s_\ell, g \rangle$ and

$$1 - D(q, g) \;=\; \frac{1}{n} \max_{|\ell| \le L} \langle s_\ell, g \rangle. \quad (2)$$

The bounded-lag top-k is therefore the top-k of a maximum over $2L{+}1$ inner-product queries against the *unshifted* stored vectors. Two consequences matter. The shift is applied to the query, so the index holds one copy of each window, not one per lag. And the probes must be inner-product queries, not cosine: $\|s_\ell\|$ shrinks with $|\ell|$ because of zero padding, exactly as the biased scorer weights a shifted overlap.

Re-ranking uses the full-precision query and the stored envelope, so every returned distance is exact; only candidate selection is approximate. For the full lag set and exact per-lag rankings with identical eligibility and consistent ties, every *unsuppressed* global top-k item occurs in the top-k of a maximizing lag. Otherwise k items would precede it at that lag and also under the maximum score, contradicting its global rank.

That lemma does not guarantee the contract's result after exclusion and overlap suppression. For example, suppose the first c candidates at every lag are overlapping windows of one recording. Suppression can retain only one; a non-overlapping window

Algorithm 1 Probe search (product function, simplified)

Require: query envelope q (full precision), eligibility predicate E, k; grid $\Lambda = \{-L, -L{+}5, \dots, L\}$; per-probe budget $c = 50$; threshold $T = 5000$

1: **if** $|\{g : E(g)\}| \le T$ **then** ▷ counted from recordings, stopping at T
2: **return** ExactScan(q, E, k)
3: **end if**
4: $C \leftarrow \emptyset$
5: **for** $\ell \in \Lambda$ **do**
6: $C \leftarrow C \cup$ top-c by $\langle \mathrm{f16}(s_\ell), g \rangle$ over g with $E(g)$ ▷ HNSW, iterative scan
7: **end for**
8: score every $g \in C$ with $D(q, g)$ on its stored envelope
9: **return** first k in order $(D, \mathrm{rec}, \mathrm{start})$ after source exclusion and overlap suppression

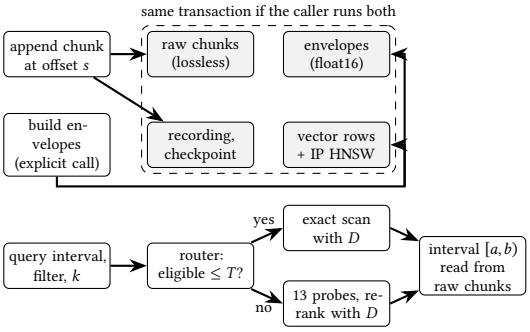


Figure 1: Write path and the two search paths. An append writes the raw chunk and advances the checkpoint; envelopes and, when probe search is enabled, their vector rows are written by an explicit builder call. All of them commit together when the caller runs append and builder in one transaction. Both search paths score candidates with D on stored envelopes and return sample intervals, which are read from the raw chunks, not from a representation. The exact path reads envelopes; the probe path reads vector rows for candidates and envelopes for re-ranking.

ranked $c + 1$ may be the second contract result yet be absent from every shortlist. Algorithm 1 also samples only 13 lags and uses approximate HNSW rankings on float16 probes. Sparse-grid loss, quantisation/index error and finite budgets interacting with exclusion/suppression must therefore be distinguished. The ablation measures the first two effects at the fixed budget; it is not a standalone budget sweep. If suppression leaves fewer than k hits, the probe function returns fewer rows while the exact path fetches more. No such short result occurred in E1.

4 Integration in PostgreSQL

MorletDB is a PostgreSQL 17 extension written in Rust with pgrx. Figure 1 shows the paths that matter for search; the supplement describes storage.

Recordings, not rows. The user-facing unit is a recording, and every search result points into one. Domain sequence types have precedent, for example the temporal types of MobilityDB and MEOS [40, 41]; our recordings use integer sample offsets and half-open intervals rather than interpolated continuous time, and their coding is bit-exact. The scorer and the envelope exist once, in a Rust core used by the extension; evaluation code re-implements them in Python, checked against the core through fixtures.

Name. The system is named in honour of Jean Morlet, after its starting hypothesis that Morlet-wavelet representations suit search in technical signal archives. Representations are versioned, replaceable extractors; the envelope used here is not a wavelet.

Recordings and appends. A collection contains recordings with a checkpoint, lossless raw chunks, envelopes and, optionally, pgvector `halfvec` rows with an inner-product HNSW index. Appends write the chunk and checkpoint atomically at an explicit sample offset. Identical retries are verified no-ops; gaps, overlaps and changed retries fail. Grizzbit stores integer-grid indices with prediction and Rice coding plus last-place corrections, or falls back to float bits; the result preserves the input float64 bits. Write-time decode verification is on by default from release 0.3.11-alpha.1. A result identifies a half-open sample interval, not a reconstructed approximation.

Storage evidence and measured versions. Grizzbit 1 was smallest on four of six waveform inputs against the B4 codec baselines; Pcodec was 5–8% smaller on the other two [49]. The pinned B6 Grizzbit 2.1 build reduced aggregate waveform bytes by 12.6% relative to Grizzbit 1 [53]. These are codec measurements, not current end-to-end database throughput. In the paired B3.8 session, MorletDB 0.3.5-alpha.1 used 33% less sample-data storage than QuestDB's Parquet arm, read short intervals faster and ingested more slowly from one connection; later B3.10–B3.12 measured the SQL 0.3.4 ingestion path separately [48]. The supplement retains the database comparisons and their qualifications. Later codec defaults, encoder fixes and releases are not silently substituted for any measured build. Figure 2 compares the codec with general-purpose lossless coders on identical samples (B1).

An append does not build search representations. The caller explicitly invokes `morlet_build_envelopes`, which also writes vector rows when probes are enabled. Samples and representations become visible together only when append and builder run in one transaction:

```
BEGIN;
SELECT morlet_append_chunk('signals', 'pump7', 81920, :x);
SELECT morlet_build_envelopes('signals', 'pump7');
COMMIT;   -- chunk, checkpoint, envelopes, vector rows
```

A builder call writes at most `max_windows` windows (10,000 by default) and returns how many it wrote; a call that hits this bound leaves later complete windows unbuilt until the next call, so a caller that needs every window covered repeats it until it returns 0. Windows of a representation may cross chunk boundaries, so a window becomes buildable only when the append that completes it has been made. Envelopes and vector rows are derived and can be dropped and rebuilt without touching raw data.

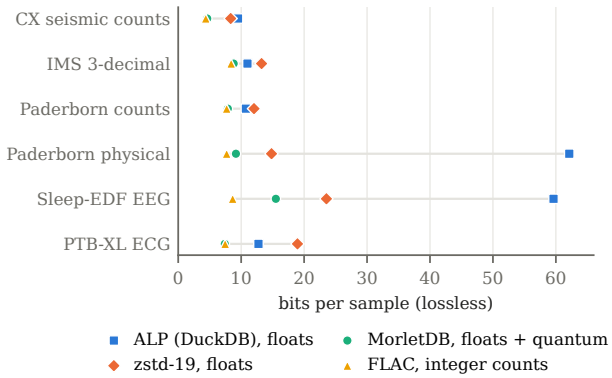


Figure 2: B1: lossless bits per sample on identical samples (float64 needs 64). Float codecs see only the float64 values; MorletDB also sees the nominal quantum; FLAC sees the integer counts. The supplement gives all arms.

Summaries, acquisitions and small machines. Optionally, each chunk also stores a 66-byte summary (count, minimum, maximum and compensated sums). Interval statistics and overviews then decode only the two edge chunks, with results bitwise identical to decoding every chunk: 58× faster at p95 for 60 s of 64 kHz vibration, and an hour of it (230 M samples) in 36 ms, for 4.5% slower ingestion and 1.3% more chunk storage [50]. Acquisitions group channels recorded together; one call reads all of them over a time interval, 32–39% faster at p50 than QuestDB for the seven channels of a B3 acquisition. Computing search features during ingestion cost 3.9× the ingest time of raw storage on the B3 data (1.64 against 6.33 M samples/s from one connection) [48]. The full workflow (ingestion with features and summaries, envelope build, bit-exact read-back of every sample, statistics and search) ran inside container limits of 2 CPUs and 512 MB at the speed it reached without limits [51], and on a Raspberry Pi 3B+ (1 GB, microSD card, no fan) at 78,000 samples/s with features and 258,000 without, returning the same search results as on the desktop [52].

Two search paths, one contract. Exact and probe search apply the same eligibility, exclusion, ordering and suppression, and both return distances computed by the same Rust scorer; a caller can switch paths without changing what a result means. The exact path scores every eligible stored envelope with D in one parallel SQL statement; the probe path implements Algorithm 1. All functions run as `SECURITY INVOKER`, so privileges and row-level security apply to every table read. The probe path finds windows through the probe table and the exact path through the envelope table, so they search the same windows only when every stored envelope has its vector row. `morlet_build_envelopes` writes envelopes and, when probe search is enabled, their vector rows in one call and continues after the last stored window, which keeps each recording's searchable windows a gap-free prefix; envelopes written by other processes, or out of order, can leave windows without vector rows (reported as `probe_backlog` by `morlet_collection_status` and filled with one `INSERT` from the envelope table) or gaps that the status does not show. The exact functions run in the snapshot of the calling statement. The probe function issues several statements; under READ COMMITTED each takes its own snapshot, so a window appended or deleted during the call may be a candidate in one statement and absent from another, while under REPEATABLE READ or SERIALIZABLE all share one. In every case each returned row was visible, eligible and scored in the re-ranking statement's snapshot. Online index rebuilds (`REINDEX CONCURRENTLY`) keep the probe path usable while they run.

Filters. An approximate index answers "nearest overall"; a filter asks "nearest among eligible". Applying eligibility after probing silently discards candidates when few windows qualify. The router therefore counts eligible windows from the recordings table, stopping at the threshold T, and sends small galleries to the exact path; above T, each probe scans only eligible recordings, using pgvector's iterative index scan. Choosing between brute force and an index by filter selectivity is established practice in vector databases (Section 11); we include it because it decides whether probe search keeps the contract under filters.

Access control. Row-level security on the recordings table alone protects the search functions but not direct reads of the derived tables. `morlet_enable_row_security` therefore attaches a policy to every companion table that admits rows whose recording is visible; written as membership in an array of visible identifiers it keeps parallel scans (exact search +5%), whereas a correlated `EXISTS` tripled exact search time. The acquisition tables, added later, first checked a channel's acquisition only by foreign key: in SQL 0.3.1 and 0.3.2, a role that could add acquisition channels could attach a recording it sees to an acquisition it does not see and then read that acquisition's metadata. SQL 0.3.3 (release 0.3.4-alpha.1) admits channel rows only into visible acquisitions, through the function and through direct writes, with a two-tenant regression test (advisory GHSA-pg6f-hcvw-vw34).

Agent access. A read-only Model Context Protocol server exposes the archive to AI agents through twelve tools: listing collections and recordings, coverage and trends over time, locating a UTC time, describing, comparing and reading intervals, finding onsets and envelope-spectrum lines, and similarity search from an interval or a time [55]. Every call runs in a read-only transaction with a statement timeout, as the configured database role, so privileges and row-level security decide what an agent sees, and every hit it cites reads back the stored samples. We report it as a feature with case studies, not as an evaluation (supplement).

5 Evaluation setup

Kinds of studies. *Confirmatory* studies committed their questions, data, arms, metrics and decision rules to the repository before their first result and ran once; deviations are logged in the protocols [5]. These are R1b, R2, BL1 and its confirmation BL1-C, BL1-S, SS1, SS3, E1, E3, E4, E5 and its held-out replication E5-H, E6, E7, B2, B3 (with its amendments B3.1–B3.12), B4, G1, CS1, EX1, EX2 and the load runs; their protocols are internal plans, not registrations with a third party. *Exploratory diagnostics* had a protocol fixed before running but no confirmatory decision attached (R2-D, D1, D2) or none at all (the pruning feasibility study, the investigation of B3.11's read difference). *Post hoc* are the grid-spacing explanation and the choice of illustrative cases;

the selection rule for the cases was committed before the figure was made, after the results were known. The supplement lists each study with its commits. CS1, EX1 and EX2 are labelled S1, X1 and X2 in the repository; we rename them because S1 and X1 name earlier studies here. Table 2 lists the studies used here.

Data. Development data (*dev*) are CX HHZ recordings [4] of stations PB01, PB02, PB06 and PSGCX, 1–14 March 2025: 481 M samples at 100 Hz, 234,762 windows of 4096 samples at stride 2048. Held-out data (*held-out*) are stations PB05, PB11, PB14 and PB18, 1–14 March 2021 (234,622 windows), chosen from the data centre's availability listing before any waveform was downloaded; no earlier study had read them. Each set has 100 queries from its first day: 50 random and 50 high-energy windows. Seismic envelopes are one signal domain; we discuss generality in Section 12.

Consolidated comparison. The main retrieval tables come from E1, a comparison of all arms on one build. E1 re-measured the held-out data that BL1-C had already used for its single confirmation (and SS3 reused), with configurations frozen before BL1-C; it is therefore a consolidated re-measurement, not another independent confirmation. It ran pg_morlet SQL version 0.3.1 at commit 68594d7 on `main`, before that version was released; release 0.3.1-alpha.1 ships it together with the later fixes named at the end of this paragraph. E1 used one configuration, the same queries and one session, under a protocol frozen before the run [31]. PostgreSQL 17.11 ran with default settings (128 MB shared buffers, 4 MB `work_mem`, two parallel workers per query; one exact arm allowed eight, of which the planner used three; PostgreSQL 17 has no asynchronous I/O), pgvector 0.8.0 (HNSW m=16, `ef_construction` 64), warm caches, one client, on an Apple M5 Max (18 cores, 128 GB) with no other CPU-intensive job (load average 3.8–4.7, mostly E1 itself). Each arm ran twice per query; latencies use all 200 timings. In both databases every envelope had its vector row, every recording's windows formed a gap-free prefix, and nothing was written during the runs, so all arms searched the same complete set of windows. Later fixes to the extension (row-security set-up, a race between routing and concurrent commits, a race between enabling probes and concurrent builds) change none of the statements measured here [31].

Earlier studies. The studies that explain design decisions (R2-D, BL1, SS1, SS3) used earlier implementation states and a machine shared with an unrelated CPU-intensive job; we label their numbers as such. Recall@10 is always against the exhaustive exact result.

6 Why zero-lag shortlists fail

A zero-lag shortlist loses a quarter of the bounded-lag top ten, and approximate search is not the reason. The natural first stage takes the top K windows by zero-lag cosine from an ANN index and re-ranks them with D. A scaling study measured this on dev galleries nested from 4,217 to 234,762 windows [7]. Its statement passed the query vector through a subquery, which pgvector cannot use as an index ordering key, so it computed an *exact* zero-lag top K. A diagnostic on the same data, with its protocol frozen before measuring, then compared an explicitly exact shortlist with an HNSW shortlist whose plan was checked to use the index.

Table 2: Studies, data and measurement kind.

Study	Question	Data
R2, R2-D	exact scaling; zero-lag shortlists, exact vs HNSW	dev
BL1	probe grids and budgets, ablation controls	dev
BL1-C	frozen configuration, once	held-out
SS3	filtered search and router; product check	dev, held-out
SS1, R1b	one stored copy; float16 precision	dev, held-out; 2022
E1	**all arms on one build**	**dev, held-out**
E3, E4	materialized exact scan outside the database; 0.23–1.03 M windows	dev, days 1–61
E6, E7	faster in-database exact scorer; screened exact search; identical results	dev, days 1–61
E5, E5-H	catalogued earthquakes, bounded vs zero lag	dev; held-out
B3, G1	storage, ingest, reads vs QuestDB, CrateDB, IoTDB, TDengine, TimescaleDB; grid detection	Paderborn; B1 inputs
B4	codec vs 2026 float codecs (Pcodec, ALP, ClickHouse PCO)	B1 inputs; Elf series
CS1	interval statistics from per-chunk summaries	Paderborn; dev
EX1, EX2	full workflow at 2 CPUs, 512 MB; on a Raspberry Pi 3B+	Paderborn
D1, D2	time per stage; filter correlation	dev
BL1-S, C2–C4	index maintenance during ingestion	dev; dev+held-out

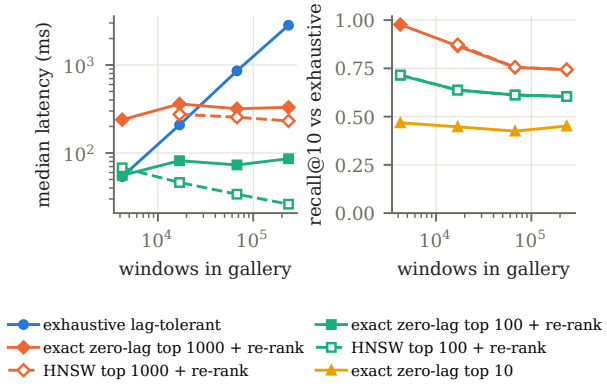


Figure 3: R2, R2-D on dev galleries. Left: median latency of the exhaustive scan (historic, single-threaded implementation) and of zero-lag shortlists plus re-ranking (exact: solid; HNSW: dashed). Right: recall@10; the HNSW curve lies on the exact one. 100 queries per gallery.

With K=1000, recall@10 fell from 0.977 on the smallest gallery to 0.743 on the largest; with K=100, from 0.714 to 0.605 (Figure 3).

Table 3: E1, held-out data (234,622 windows, 100 queries, 200 timings per arm). Recall@10: mean / minimum / queries below 0.9.

Arm	Recall@10	p50 / p95 ms	Work per query
exact, 2 workers	1	958 / 971	score all windows
exact, 8 allowed (3 used)	1	728 / 740	score all windows
probes (product)	**0.977 / 0.7 / 5**	39 / **57**	13 scans, re-rank ≈350
zero-lag HNSW, 650	0.639 / 0.1 / 69	28 / 40	1 scan, re-rank 650
zero-lag HNSW, 1000	0.658 / 0.1 / 67	34 / 50	1 scan, re-rank 1000

The HNSW shortlist agreed with the exact one on 98–100% of candidates on the largest gallery and gave the same recall to within 0.007 at every size (0.744 at K=1000). Every top-ten item contained in a shortlist survived re-ranking and suppression, so coverage equalled recall. The zero-lag candidate metric causes the loss, not approximate search and not the contract's exclusion rules: a quarter of the bounded-lag top ten lies outside the 1000 windows nearest by zero-lag cosine. The exact zero-lag top ten alone shared 43–47% of the bounded-lag top ten. E1 reproduced this on its pinned build: a zero-lag HNSW shortlist of 1000 from the probe index recovered 0.744 on dev and 0.658 on held-out data. The exhaustive scan in Figure 3 is the historic implementation (linear, 12.1–12.8 µs per window); the E1 exact path runs as one parallel statement and is the baseline in Table 3; E7 measures the later screened path separately.

7 Shifted probes: ablation and confirmation

The configuration was chosen on dev data [10]: grids of 7 and 13 probes and budgets of 50 and 100 per probe all met its gate (recall@10 $\geq$ 0.95, p95 at most a third of the exhaustive scan and below the fastest exact path, added storage at most 1.5× raw); 13 probes × 50 was frozen and confirmed once on held-out data. E1 then measured the product functions and all controls on one build [31]. Its offline controls use the same stored float16 envelopes and the same re-ranking: *PX-61*, the exact top 50 for each of the 61 lags; *PX-13* and *PX-7*, the same for the 13- and 7-lag grids; *PQ-13*, as PX-13 with each probe rounded to float16, as the index sees it; and *EZ*, an exact zero-lag shortlist with as many candidates as PX-13.

Benefit. On held-out data probe search kept 97.7% of the exact top ten at a p95 latency of 57 ms, against 740 ms for in-database exact search on the measured build (Table 3); on dev data, 99.8% at 53 against 783 ms. Zero-lag candidates from the same index recovered 64–66%, even with a larger re-rank budget; an exact zero-lag shortlist as large as the probes' candidate set recovered 61%. By query group, held-out recall was 0.998 for the 50 random windows and 0.956 for the 50 high-energy windows; energy is a selection rule for queries, not a measure of relevance. Of the 100 held-out queries, 83/12/4/1 reached 1.0/0.9/0.8/0.7. The 23 missed intervals (of 1,000) ranged from rank 1 to 10 in the exact list and 3 were the exact best match; lower-ranked admissible intervals took their place. Both rounds on the same index returned identical lists; a rebuilt HNSW index can return different candidates.

Against a faster exact path. On the measured build, 97% of exact search time was spent in the per-row scorer. We rewrote it to build the query's shifted copies once per query and to score each window in one pass across lags, keeping every lag's summation order, so distances stay bitwise identical; the exact lists equalled the measured build's for 100 of 100 queries at each gallery size [45]. In-database exact search became 1.6–2.7 times faster at p95. With eight workers its p95 was 277, 575 and 1,551 ms on dev galleries of 0.23, 0.50 and 1.03 M windows, against 55, 68 and 238 ms for probe search: probes stayed 5.1–8.5 times faster, with every bootstrap interval above 1.

Against a screened exact path. Most windows cannot be among the nearest, so the E7 product build screens them first [46]. A parallel aggregate over the envelope rows scores every window in float32 against the shifted query and bounds the error rigorously (rounding of the dot products and of the window's mean and spread); only windows whose lower bound is at most the m-th smallest upper bound are re-scored with the unchanged float64 scorer, and the contract's selection rules follow as before. The lists equalled the measured build's and the unscreened path's, bit for bit, for 100 of 100 queries at each gallery size. In one session with eight workers, the screened path's p95 was 138, 221 and 540 ms on 0.23, 0.50 and 1.03 M windows, against 516, 808 and 1,928 ms for scoring every window (3.6–3.7 times faster; 3.2–3.6 with the default worker count) and 88, 130 and 237 ms for probe search: probes stayed 1.6–2.3 times faster at p95 in every configuration, with every bootstrap interval above 1. At 0.23 M windows the median screened search took 77 ms against 64 ms for probes. The machine was busier than in E6, which slowed both paths, so only these within-session ratios are comparisons (Figure 4).

Against a materialized exact scan. Outside the database, exact search can be faster still [44, 61]. All envelopes are held in memory as a float32 matrix; one matrix product scores all 61 lags, a rigorous rounding bound selects the windows to re-score in float64, and the contract's selection rules are applied in the same order. This scan returned exactly the in-database result for every query. Its p95 was 21, 42 and 80 ms on galleries of 0.23, 0.50 and 1.03 M windows, against 68, 108 and 202 ms for probe search at recall 0.99–0.998; no crossover was measured, and fits place one near 34 M windows. The scan holds a 5 GB copy at one million windows, decoded once and scored with the processor's matrix unit, outside the database and without its filters, row-level security or commit-time visibility. Inside that contract, screening brings the same bound into the database, but per-row work remains (reading each row and computing each window's mean and spread at query time): the screened path was 3.5–5.6 times slower than this scan at p95, and probes are the faster path.

Where the loss comes from. Table 4 separates the approximations of Section 3. Each control keeps the 50-row budget per probe and the stored float16 envelopes. On held-out data, exact candidate generation on the sparse grid recovered 0.999 of reference intervals (99.9%); rounding the probes to float16 introduced no further observed loss (0.999). HNSW candidate generation reduced recall to 0.977. Most observed loss therefore arose from approximate candidate selection, with one reference interval already

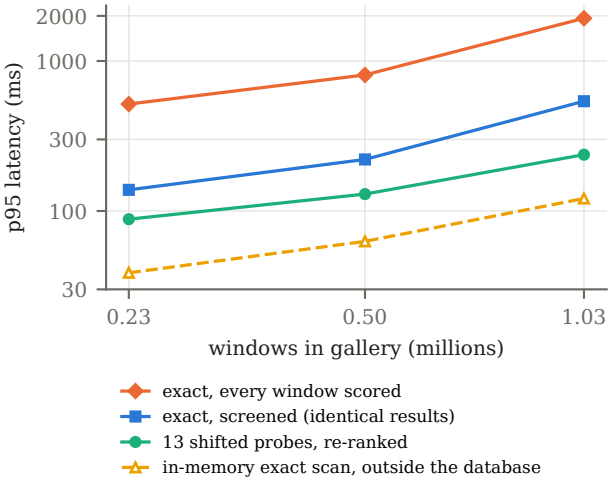


Figure 4: E7, one session, eight workers, dev galleries: p95 of 200 timings per arm. Screened exact search returns exactly the results of scoring every window; probes recalled 0.99–0.998 of them. The in-memory scan runs outside the database, without filters, row security or commit-time visibility.

Table 4: E1: recall@10 by candidate generator. All arms re-rank with D.

Candidates	dev	held-out
all 61 lags, exact (PX-61)	1.000	1.000
13-lag grid, exact (PX-13)	1.000	0.999
13-lag grid, float16 probes, exact (PQ-13)	1.000	0.999
7-lag grid, exact (PX-7)	0.986	0.987
13-lag grid, HNSW (product)	**0.998**	**0.977**
zero-lag, exact, same count (EZ)	0.681	0.613

missed by the sparse grid. On dev data the grid lost nothing and HNSW 0.002. A grid at spacing ten lost 0.013–0.014. HNSW with full-precision probes cannot be measured separately, because the `halfvec` index takes float16 queries. Missing probe rows, differences between searched windows and concurrent writes are separate possible sources of loss; they were excluded here by construction and by the checks above. None of this guarantees the same split on other data or with other HNSW settings. The six hardest held-out queries are all high-energy windows of one station, and each reaches 1.0 with exact grid candidates. Figure 5 shows the mechanism: zero-lag candidates find items whose best lag is within ±5 but almost none beyond ±15, whereas probes are flat across lags. Figure 6 shows one case of each kind.

Why spacing five sufficed here. For each of the 1,000 reference items per database, the best of the 13 grid probes kept a median 0.999 of the item's best-lag correlation (5th percentile 0.98, minimum 0.84), and ranked the item at median 2 and 95th percentile 7–8 among all windows in that probe's exact ordering; one held-out item ranked 324th, outside the 50-row budget, which is the single PX-13 miss. On 3,000 dev envelopes the envelope's own

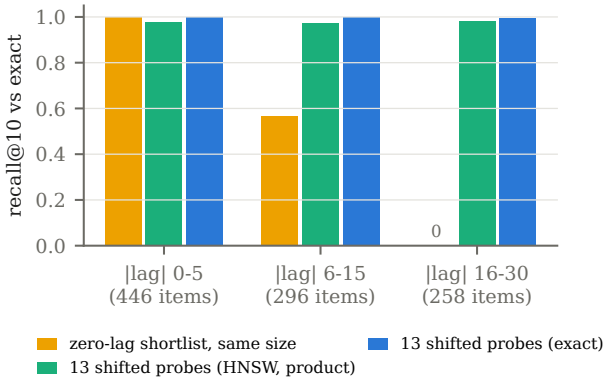


Figure 5: E1, held-out: recall of the exact top ten by the lag at which each item scores best.

biased autocorrelation is 0.973 at an offset of two values and 0.895 at five [43], consistent with the smoothing applied before decimation (about five envelope values). This explains why spacing five works on these data. Recall is decided by the rank inside a probe, so for a new signal the spacing is a measurement, not a consequence of smoothness alone.

Where the time goes. An instrumented copy of the probe function on dev data [31] spent 65% of its time in the 13 index statements (1.9 ms each, p50) and the rest in re-ranking, reading the source interval and routing; instrumentation added about 13 ms to the product's 29 ms, mostly by materializing the re-rank, so the index statements account for most of the product's time. The eligibility predicate cost 0.03 ms per statement. Issued on its own, one index statement took 0.38 ms to plan and 0.82 ms to execute (`EXPLAIN ANALYZE`); inside the function each probe statement took about 1.9 ms. Planning inside the function was not measured, so the remaining 0.7 ms per probe is unresolved; planning of the dynamically issued statement, building the probe vector and collecting the result are candidates. ANN traversal, heap access and visibility checks happen inside one index-scan node and were not separated. Every probe returned 50 rows; the union held 335 windows (median), one was excluded as the source, none was suppressed.

8 Filters and routing

Investigations usually restrict the gallery to one sensor or period. The first probe implementation applied eligibility after probing. A filtered-search study compared three strategies at five filter levels per query [12]: exact search, post-filtered probes and filter-aware probes; a router threshold was fixed on dev data before the held-out run.

Post-filtering lost up to half the top ten (Table 5). Filter-aware probes restored recall but were slow at the smallest level (p95 724 ms), where the planner abandoned the index and exact search took 68 ms. The router sends that level to exact search. No returned row violated its filter, and with row-level security restricting a role to one station, every strategy returned only that station and rejected a hidden recording as query source. The router missed its pre-registered latency bound once, at the unfiltered held-out level,

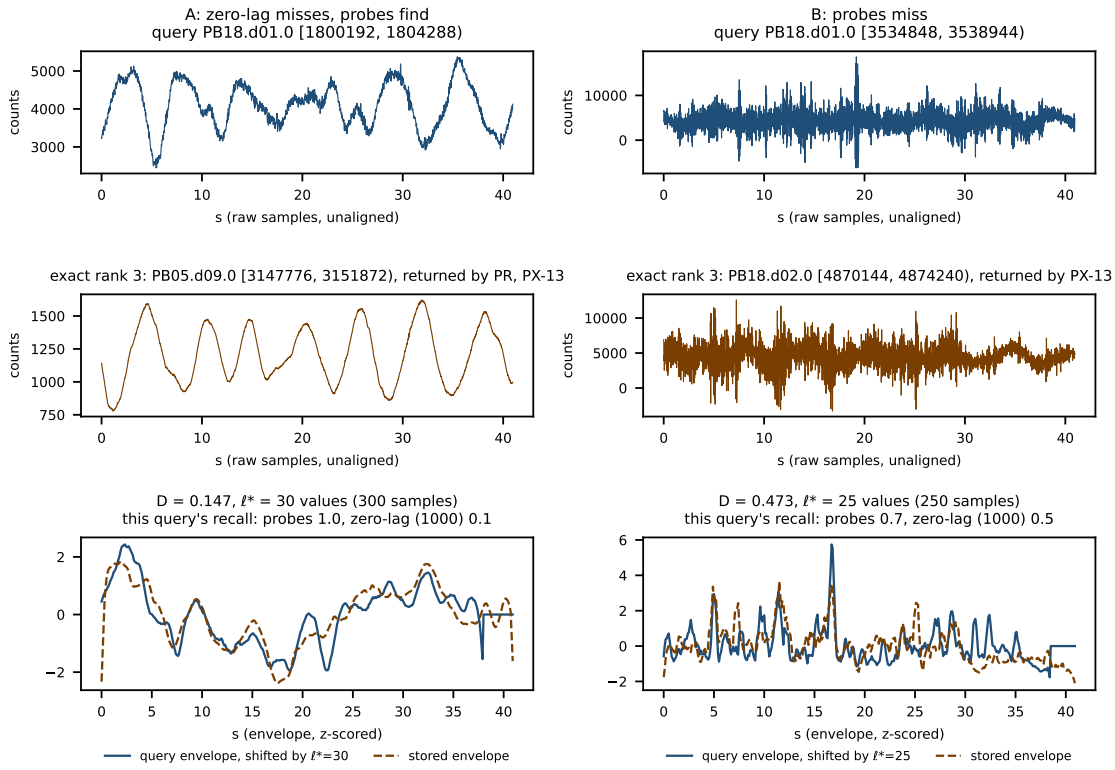


Figure 6: Two held-out cases, selected by a rule fixed before the figure was made: (A) the query with the largest recall gain of probes over a zero-lag shortlist of 1000, and the reference interval with the largest best lag among those the probes find and the shortlist misses; (B) the query with the lowest probe recall, and its best-ranked missed interval, which exact grid candidates (PX-13) do find. They illustrate, and do not replace, the distribution in Table 3; no domain label is implied. Top rows: raw samples in counts, unaligned. Bottom: z-scored envelopes as the scorer uses them, the query shifted by the best lag ℓ^* with zero padding; the distance D is the product scorer's and was checked against $1 - \max_\ell c(\ell)/n$.

Table 5: SS3 (earlier build, shared machine): filtered search on held-out data, mean recall@10 (product: also minimum) against exact search, 100 queries per level. Post-f.: post-filtered probes; F-aware: filter-aware probes; Routed: product function (*routed to exact).

Eligible	Exact p95	Post-f.	F-aware	Routed mean/min	Routed p95
1.8% (4,217)	68 ms	0.515	0.998	**1.0**/1.0*	85 ms
7.2% (16,868)	329 ms	0.848	0.997	**0.997**/0.9	207 ms
25% (58.6 k)	290 ms	0.972	0.986	**0.986**/0.6	98 ms
29% (67.5 k)	390 ms	0.976	0.994	**0.994**/0.8	88 ms
100% (234.6 k)	~1.2 s	0.976	0.977	**0.977**/0.7	69 ms

by 0.8 ms. Low per-query minima at 25% and 100% coincide with the high-energy queries of Section 7, not with routing errors.

Correlated filters. A filter can keep the same number of windows but more or fewer of a query's neighbours, which changes how far an index scan must travel to find eligible rows [38]. A diagnostic chose, per dev query, the day filter (about 7% of windows) or station filter (25%) that held the most, a random, or the fewest of the query's unfiltered top hundred; the choice is synthetic [31]. Recall was 0.997–1.0 in every case. Latency was not: with day filters, probe p95 rose from 121 ms (positive) to 178 (neutral) and 227 ms (negative), against 210 ms for exact search on the now quiet machine; with station filters, probes stayed at 58–63 ms against 264 ms. Routing by the number of eligible windows therefore preserved recall in this workload, but its latency crossover depends on how the filter relates to the query, and the threshold, set on a shared machine, sends 7%-filters to probes where exact search is as fast.

9 Storage, recall and latency

Probe search costs space. In E1, raw chunks took 371 MB (dev) and 514 MB (held-out), envelopes 223 MB (read by both paths for scoring), the probe table 223 MB and its HNSW index 275 MB: the probe structures added 1.34× and 0.97× the raw data. Rebuilding the index

took 39–41 s with 4 GB of maintenance memory. Smaller storage and faster search are separate goals: float16 probes did not change recall, and storing fewer bytes need not make search faster. How large envelopes are relative to raw data depends on stride and on how well the raw data compress (supplement).

Float16 storage is not the bottleneck: on 2022 windows that no earlier study had read, float16 envelopes kept recall@10 at 0.9999 against envelopes recomputed from raw samples [6]. The two stored copies hold the same float16 values in 98.3% of positions. A study of keeping only one [11] found that a single `halfvec` table was exact on both databases (identical distances for 9.4 M pairs) and saved about 250 MB per 235,000 windows, but made exact scans 23% slower; an HNSW index on an expression over the envelope table kept exact search unchanged but raised probe p95 by 27–32%. Both missed their pre-registered latency bounds. The historical SS1/E1 trade-off is three-way: probe structures cost about 1–1.3× raw storage, and eliminating a stored copy increased latency by 20–30% on one path. Against the later screened exact implementation, the within-session probe advantage was 1.6–2.3× on development galleries (Section 7); the older latency ratios are not claims about the current release.

Maintenance during ingestion. Building vector rows at commit makes new windows searchable immediately; the price is HNSW maintenance on every append. In a replay of one day of four channels, appends ran at 53% of raw-only throughput with every window searchable at its commit [10]. Under sustained load (409,600 samples/s with concurrent searches on 938,768 windows), write-ahead logging was 18 times the database growth with PostgreSQL's defaults and 2.9 times with four settings changed together; a planned two-hour run met its target for 111 minutes and stopped after an unexplained pause [13]. The supplement reports these runs.

10 Industrial use

Condition monitoring motivated MorletDB, but the retrieval results above are seismic. This section reports two studies on public machine data, CNC milling runs labelled good or bad [57] and milling tools run to failure [58], and how fast their queries run under edge-sized limits [59].

Study. We used the Bosch CNC milling data [56] (CC BY 4.0): 1,702 runs of 15 operations on three machines, labelled good or bad per run, tri-axial acceleration in integer counts at 2 kHz, 178.6 M samples per axis. The data give the half-year of a run, not its time. Each run and axis is one recording; envelopes use the product defaults (windows of 4,096 samples, stride 2,048, L from `max_lag_fraction` 0.075). Machine M01 served to check the harness; M02 and M03 (1,183 runs, 36 bad) are the confirmation data. The protocol, with every decision rule, was frozen before loading. Up to eight query windows per run and axis; hits from the query's own run are removed. We compare bounded lag (BL) with the same scorer at zero lag (ZL) and with nearest windows by RMS, and decide by a 95% run bootstrap.

Scoring runs against a good history. Scoring every run by its distance to the nearest window of good runs of the same machine, operation and axis gave an AUROC for bad against good of 0.925 (95% interval 0.878–0.967) under BL, 0.875 under ZL and

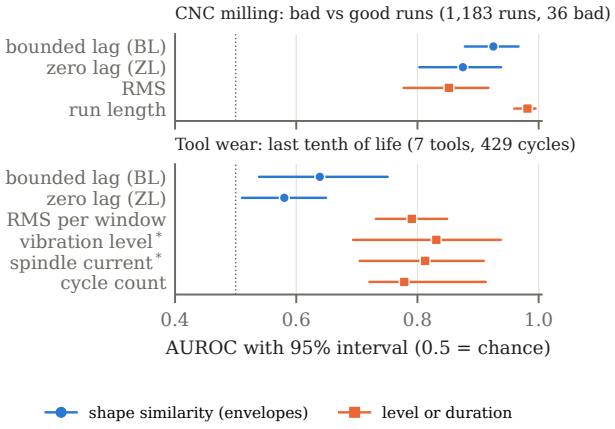


Figure 7: IU1 and IU2 confirmation data: how well each score separates bad from good runs (top) and the last tenth of a tool's life from earlier cycles (bottom). Run bootstrap (IU1) and tool bootstrap (IU2) intervals. Shape scores come from MorletDB's envelopes; RMS per window from MorletDB's interval statistics; *from the dataset authors' feature table, not from MorletDB.

0.852 under RMS. BL was better than ZL; against RMS the difference was not significant. BL and ZL shared only 38% of their hits. The originally reported cross-machine recognition result (0.503 under BL against 0.412 under ZL) is withdrawn: its harness removed source-machine hits after a fixed shortlist and used a variable denominator. A post hoc rerun that excludes the source machine before ranking and divides by ten found the same operation on another machine for 0.523 of the top ten under BL and 0.412 under ZL (chance 0.076; difference 0.110, 95% interval 0.077–0.145, and 0.075–0.147 when resampling the 187 source runs). The corrected search was written after the original results had been seen, so this is an exploratory result, not a confirmation.

Is it drift? Twenty of the 36 bad runs come from one machine and half-year, so after seeing the results we checked, with rules written down first, whether the score measures drift between half-years. It does not: with the history restricted to the query's own half-year the AUROC was 0.933 (0.895–0.965), and over pairs of a bad and a good run from the same machine and half-year 0.905 (0.853–0.950) under BL against 0.790 under ZL (difference +0.116, 0.047–0.184). Without that machine and half-year it was 0.854 (0.768–0.932), and without each run's first two seconds 0.900 (0.836–0.954), so the lead-in does not carry it either. Three restrictions follow. A history from earlier half-years only did not separate the classes (0.603, 0.419–0.780, 13 bad runs): the signal changes between half-years by more than a bad run differs from a good one, so the history must be recent, and the data do not allow a causal test at finer resolution. Run length alone separated better than any signal score (0.982 against 0.925; inside a machine and half-year 0.988 against 0.905): in these data a bad run is mostly a run of unusual length. And retrieval of other bad runs was inflated by the half-year: inside the query's own half-year the top ten held

bad runs in 0.254 of positions under BL, 0.212 under ZL and 0.377 under RMS, at a chance level of 0.183 (the frozen protocol had reported 0.177 against a chance level of 0.026).

Time warping. We stretched query intervals by 1, 2, 5 and 10% to imitate a feed override and searched for the source interval. It stayed in the BL top ten for 0.97, 0.76, 0.51 and 0.20 of 100 queries, against 1.00, 0.79, 0.20 and 0.07 for ZL. Bounded lag absorbs small offsets, not changes of speed.

Probes. On an unfiltered gallery of 85,042 windows, probe search returned the exact top ten for all 100 queries, at a p50 of 56 ms against 141 ms for exact search and a p95 of 126 against 176 ms. The frozen rule asked for half the exact p95, so the speed claim does not hold at this gallery size; it agrees with Section 12 that exact search suits galleries below a few hundred thousand windows.

Tool wear. A second study used 13 end mills run to failure [60] (CC BY): 967 machining cycles, three spindle accelerometers at 25 kHz, 4.88 billion samples, with tools 2, 3, 7, 8, 101 and 102 for development and seven tools for confirmation [58]. Its protocol was frozen before the raw signals were read. Each cycle was scored by its distance to its own tool's first cycle at the same position, and the end of life (the last tenth of a tool's cycles, 48 against 381 earlier) was detected across the confirmation tools with an AUROC of 0.639 (0.539–0.751) under BL and 0.580 under ZL; the difference was not significant here, while on the development tools BL was better (+0.023, 0.010–0.054). Level values did better than either: RMS per window (`morlet_interval_stats`) 0.791, and, from the dataset authors' feature table rather than from MorletDB, the vibration level 0.831 and the spindle current 0.813; counting cycles gave 0.778. The reason shows in the distances: a new tool's cycles were already 0.45 from their own baseline, and wear moved them only to 0.50. Steady milling repeats little envelope structure from cycle to cycle; what wear changes is the level, which the normalised envelope removes. With one threshold from the development tools, RMS per window flagged all seven confirmation tools in a cycle strictly before failure, and vibration level six of seven; vibration level first flagged the seventh in its breaking cycle, which is not advance warning. These scores also flagged 12–20% of cycles in the first half of life. Bounded lag warned on one tool before failure and first flagged another in its breaking cycle. The 4.88 billion samples took 6.47 GB of chunks (10.6 bits per sample) against 39.1 GB as float64.

Under edge-sized limits. The same queries ran with the database container unlimited and limited to 2 CPUs and 512 MB without swap, as in EX1 [59]. A search in the good history of one operation (about 2,000 windows) took 32 / 42 ms (p50 / p95) under the limits and 32 / 43 ms without; statistics of one window 1.1 / 1.6 ms; a decision per run from 24 such searches 726 / 878 ms. Only the search over all 85,042 windows slowed down, from 212 to 366 ms at p95 for exact search and from 173 to 197 ms for probes, because fewer workers were available. No query ran out of memory (peak 404 MB, mostly page cache). These are laptop cores under container limits, not edge hardware, with one client and warm caches.

What follows for industrial use. Lag tolerance improves envelope similarity on machining vibration as it does on the tested classification comparisons: BL had higher AUROC than ZL, significantly on the Bosch data but not on the seven confirmation tools of IU2. It was not uniformly better: ZL recovered more sources under 1% and 2% synthetic stretching. A nearest-neighbour score against a recent good history separates bad runs without training, but simpler values do as well or better on both datasets: RMS for retrieval, run length for scoring bad runs, and level values for tool wear. The envelope is normalised and so blind to level and duration; a product should offer both next to the shape, and the level values that a decision needs per part take milliseconds where a decision from shape searches takes most of a second. A changed speed needs alignment or segmentation by NC block before search; at 5% stretch half of the sources are lost. The studies cover two public datasets, run-level labels without time stamps (Bosch) and cycles to failure without wear measurement (13 tools, seven for confirmation), single axes or product-default envelopes not tuned to the tooth-passing frequency, and synthetic stretching, on one laptop; they make no claim on diagnosis or on real-time decisions at the machine.

11 Related work and what is new

We looked for the strongest overlapping prior work, reading primary sources; Table 6 records for each work whether we read the full text (F), parts (P) or only the abstract (A). The full matrix, with quotations and what we searched for, is in the repository [5]. We use *shift* for a displacement along the time axis. Invariance to a vertical offset (adding a constant), as in normalization-based methods [29], and shift-invariant kernels that average over all circular shifts [19] are different notions.

What is known. Three ideas we use are established. (i) *Shifting the query instead of the data.* Rafiei [14] evaluates similarity under a set of transformations, explicitly including time shifts, against one index of untransformed sequences, and processes all transformations in one index traversal; Keogh et al. [15] expand a query into all its rotations, keep the database stored once, support rotation-limited queries by dropping rows of that matrix, and merge neighbouring rotations into a few envelopes, an exact counterpart of our sparse grid. Foulds et al. [17] search under cyclic permutations of the query. All three are exact, use Euclidean-type distances, and treat shifts as circular. (ii) *Several vector queries against one approximate index, union, exact re-ranking.* ColBERT's end-to-end retrieval issues one ANN search per query vector, takes the union of document identifiers and re-ranks it exhaustively [24]. (iii) *ANN candidates for cross-correlation.* In seismology, Tibi et al. [22] project waveforms with kernel PCA so that correlation maps to Euclidean distance, search a KD-tree forest and evaluate exact correlation on the returned set (25× faster than linear search at recall above 80%); Gonzales et al. [23] issue several internal queries to compensate for projection error. FAST obtains shift tolerance from densely overlapping data windows [2, 21], and GRAIL learns embeddings of a shift-invariant correlation kernel [19]. The identity between a lag term and an inner product with a shifted signal is textbook; the biased statistic itself appears as NCC_b in the k-Shape work [18]. On the database side, choosing brute force over an index when a filter is selective is standard in vector systems (AnalyticDB-V, Milvus, ACORN, and the documented thresholds of Weaviate, Qdrant, Vespa and Lucene) [25–27]; VBase and pgvector's iterative scans provide filtered index scans inside

Table 6: Closest prior work. Shift: which displacements along the time axis the similarity tolerates. Side: where the shift is applied. Cands.: whether the candidate set is exact for the stated similarity. Read: F full text, P parts, A abstract. [a]exact for the magnitude feature, which itself approximates the rotation distance.

Work	Similarity	Shift	Side	Index	Cands.	Copies, refinement	Read
Rafiei 1999 [14]	Euclidean on normal form, truncated DFT	set of transformations incl. time shifts (circular in DFT)	transformations applied to index entries	R-tree on DFT coefficients, one traversal for all	exact	one copy; false positives removed exactly	F
Keogh et al. 2006/09 [15]	Euclidean (also DTW)	all circular rotations or a limited range	query: matrix of rotations, merged into wedges	lower bounds; VP-tree	exact	one copy; exact refinement	F
Vlachos et al. 2005 [16]	Euclidean on Fourier magnitudes	all circular	data: invariant feature	VP-tree variant	exact[a]	one compressed signature	P
Foulds et al. 2015 [17]	distance over cyclic permutations of the query	all cyclic	query	none; parallel DFT-based search	exact	—	A
k-Shape, TODS 2017 [8, 18]	normalized CC incl. biased variant	all linear shifts (FFT)	pairwise	none	exact	—	P
GRAIL 2019 [19]	kernel over all circular shifts of NCC	all circular	data: learned embedding	any (Euclidean/IP)	approx.	one embedding per series	P
Qiu et al. 2018 [20]	Pearson at zero lag	none	data: learned embedding	Euclidean index	approx.	one embedding	P
MASS, UCR suite [3, 9]	z-normalized ED per subsequence	every offset	scan	none	exact	—	A
FAST, Rong et al. [2, 21]	Jaccard on binary fingerprints	data windows every 1 s	data: dense windows	MinHash LSH	approx.	one fingerprint per overlapping window	F
Tibi et al. 2017 [22, 23]	max-over-lag CC of waveforms	bounded lags	data: kernel-PCA embedding of CC	KD-tree forest	approx.	projected vectors; exact CC on candidates	F
Kwon et al. 2007; Moirae [34, 35]	context similarity of events over monitored values (CDM)	none	—	history partitioned by time, recent first	approx., refined incrementally	scalar snapshots in PostgreSQL; no waveforms	F; P
ColBERT 2020 [24]	sum of max inner products	—	query: one vector per token	one ANN index, one search per vector	approx.	union of candidates re-ranked exactly	F
This paper	biased, zero-padded CC	linear, $\lvert\ell\rvert \le L$	query: sparse grid of shifted copies	unmodified IP HNSW, one search per probe	approx.	one vector copy + index; union re-ranked with D	—

PostgreSQL [1, 36]; SingleStore-V makes vector search visible at commit by indexing immutable segments and scanning fresh rows exactly [28]; and KV-match and ULISSE return exact subsequence matches as offsets in stored series [29, 30].

Vector search inside PostgreSQL. VBASE exposes an index as an iterator that the executor can resume until enough rows pass the filters, and stops when the traversal moves away from the query [36]; its guarantee is equivalence to a top-K-then-filter plan with the best K, which needs filters that commute with sorting. Our score is a maximum over lags of which only the grid is probed, and greedy overlap suppression does not commute with sorting, so that argument does not carry over; pgvector's iterative scan already resumes within each probe statement, and E1 and its diagnostics found no query whose fixed budget ran out. PostgreSQL-V moves index contents out of PostgreSQL pages and indexes recent rows asynchronously, giving up the WAL-logged, commit-time index visibility that our contract uses [37]. A study of filtered vector search in a PostgreSQL-compatible engine shows that correlation between filter and query neighbourhood, not only selectivity, governs the cost of iterative scans [38]; D2 observed the same effect on latency, not on recall. MorletDB uses no PostgreSQL hooks and no copied internals, the patterns that a survey of extensibility associates with incompatibilities; its dependencies are PostgreSQL 17 and pgvector 0.8 or later [39].

Compressed high-frequency sensor data. ModelarDB [32, 33] manages high-frequency sensor time series across edge, cloud and client: it compresses each value column on the edge with constant and linear models and stores the points no model covers as residuals, either lossless or within a per-column error bound; it ships the compressed segments to object storage and answers SQL over Apache Arrow, DataFusion and Parquet, computing some aggregates from the models. Its evaluations use wind turbine data sampled every 2 s and every 150 ms. The overlap with MorletDB is compression, SQL access and the edge; the difference is the unit of storage and what a query returns. ModelarDB stores 32-bit floats, and its lossless setting guarantees equal values rather than identical bits; MorletDB stores recordings of 64-bit samples bit for bit in PostgreSQL, keeps each recording's acquisition context, and returns the original samples of similar intervals, which ModelarDB does not search for. Its model-based compression targets smooth signals and error-bounded storage, MorletDB's codec dense waveforms that must stay exact; a measured comparison on both kinds of signal is future work.

Similar past events in monitoring. Moirae [34] couples a stream processing engine with PostgreSQL as the historical log and, for each newly detected event, returns the k past events with the most similar context, approximately at first and refined incrementally, so that an administrator sees earlier occurrences alongside the alert; Kwon et al. [35] study the similarity measure for this setting. Their Context Distance Measure compares an event together with its context, the state of the monitored objects around

it, and ranks past events highest when the same objects show similar abnormal values; on PlanetLab monitoring data it ranks better than the Jaccard coefficient, the Earth Mover's Distance and Match-and-Compare. They compare scalar snapshots: there are no waveforms, no tolerance to time shifts and no exact read-back of the evidence. Their premise, that context decides which past cases are comparable, is the one the decision state builds on: it compares a segment only with segments of the same context (match keys), and the industrial studies report their results per context (§10). The claim below is not affected: these works do not search waveform intervals under shifts.

What is new here. Query-side shifting, multi-query candidate generation and selectivity routing are each known. To our knowledge, no prior work combines them for a *bounded, zero-padded, linear* lag scorer on an *unmodified, approximate inner-product index*. Equation 2 lets a bounded-lag score use any inner-product ANN index without training an embedding (unlike Tibi et al. and GRAIL), without replicating windows per shift (unlike FAST) and without lower bounds (unlike Rafiei and Keogh et al.); in exchange, candidate selection is approximate, and we measure its loss. The diagnosis that zero-lag candidates fail through their metric rather than through approximate search, and the ablation that attributes the remaining loss, are to our knowledge new as measurements. On the systems side, MorletDB keeps intervals, filters and row-level security identical on the exact and approximate paths, and makes built windows searchable at the commit that builds them.

The single-traversal alternative. Rafiei and Keogh et al. show that processing all transformations in one exact traversal with lower bounds can beat one search per transformation. Before building probe search we examined the two lag-uniform bounds closest to this design on our scorer [42]: a Keogh-style envelope bound (one wedge over all lags $|\ell| \leq L$) and a spectral bound, on 1,826 windows of 2022 data. This was a feasibility study: even under idealized assumptions (the true tenth-best distance as threshold, bounds computed at no cost) their combination pruned 63% of candidates, because it overestimated the best correlation by a median factor of 1.83. Inserted into the measured per-candidate costs of the exact scan of that time, this gives a modelled speed-up of about 1.9× on that gallery and 2.3× in the limit of large galleries. These are model values for one construction, not measurements; finer wedges or other bounds could do better. They explain why we pursued approximate candidates rather than exact pruning.

12 Limitations

Scope of the claims. We do not claim a new similarity measure or a new index, and probe search has no recall guarantee: its recall is measured, not bounded. Recall is fidelity to the scorer, not relevance to a domain question.

One signal domain. All retrieval results outside Section 10 are for seismic envelopes from one network, and our empirical claims are limited to these data and, for machining vibration, to the two public datasets of that section. The method needs only a bounded-lag inner-product scorer, but how many probes and candidates suffice depends on the representation's smoothness and on where best lags fall relative to the grid; apart from one gallery of 85,042 machining windows (Section 10), we have not measured this on vibration, acoustic or physiological data. Condition monitoring motivated the work and is a target application; beyond the run-level labels of one public dataset, we have no evidence that probe search, or envelope similarity in general, is diagnostically relevant for machine condition. On those data, level values (RMS) and run length detected bad runs, and level values tool wear, as well as or better than envelope similarity, and on the Bosch data the good history had to be recent; the data give only the half-year of a run, so a causal test at finer resolution was not possible.

Earlier implementation states. E1 measured the main comparison on one build. The filtered-search table (SS3), the single-copy study (SS1), the zero-lag scaling study (R2, R2-D) and the load runs were measured on earlier states and on a machine shared with another job; ratios across those studies mix states.

Speed relative to exact search. The speed-up is against in-database exact search with results identical to the product's. Held-out speed-ups (Table 3) are against the measured build; against the screened exact path measured in E7, probes were 1.6–2.3× faster at p95 on dev galleries up to 1.03 M windows. A materialized exact scan outside the database was 2.5–3.3× faster than probe search at p95 up to 1.03 M windows, without the contract.

Measurement conditions. One laptop-class machine, warm caches, one HNSW configuration; only EX2 and the industrial studies ran on other hardware: EX2 measured ingestion, reads and search speed, not retrieval quality, and IU1, IU2 and IU1-E ran on another laptop under Docker, IU1-E also with the container limited to 2 CPUs and 512 MB. The load tests ran 15 and 111 minutes; the planned two-hour run stopped after an unexplained pause, and we make no claim about longer operation, archives beyond about one million windows, cold caches or several tenants under load.

What recall means. Recall@10 measures fidelity to the scorer D. Whether the bounded-lag top ten are the intervals an analyst wants is a separate question. A proxy task gives first evidence for the scorer: with a query window containing a catalogued earthquake's predicted P arrival, the exact bounded-lag top ten contained such windows in 0.297 of positions against 0.213 for zero lag on development data (465 queries from 171 EMSC events; difference +0.084, 95% interval $[0.070, 0.098]$ by event bootstrap), and 0.139 against 0.060 on held-out stations and year (116 queries from 92 events; +0.078, $[0.054, 0.105]$), against a chance level below 0.005; both frozen decision rules read "bounded lag better" [47]. Catalogue membership is a proxy for relevance, the absolute precision is modest, the comparison used exact search only, and we have no task evaluation of probe search; given its recall, we expect but did not measure a similar gap.

Approximation without bounds. Probe search can miss a window that the exact path returns (per query, down to 7 of 10 on held-out data); the budget is fixed and is not extended when exclusion or suppression removes candidates. The idealized lower-bound study of Section 11 covered one construction; finer bounds were not tested.

Routing. The router's threshold counts eligible windows only. The correlated-filter diagnostic showed that its latency crossover depends on how the filter relates to the query and on machine load; we did not recalibrate it.

Grid choice. The 13-probe grid was chosen on dev data and explained post hoc by envelope smoothness; it was confirmed on held-out data, for this representation only.

When to use it. Exact search for small galleries (a few thousand eligible windows), for selective filters that exclude the query's neighbourhood, and whenever every true neighbour must be found. With screening, exact search took 0.14–0.54 s at p95 on dev galleries of 0.23–1.03 M windows, so it is an interactive option up to about a million windows when every true neighbour matters. Probe search for unfiltered or weakly filtered galleries of a quarter of a million windows and more, where it was 1.6–2.3 times faster at p95 than the screened exact path at recall 0.99–0.998 (Section 7), and for larger archives, since exact search reads every eligible window. When a separate in-memory copy without filters or security is acceptable, a materialized exact scan is faster still. The crossover points measured here hold for these data and this machine.

13 Conclusion

Sensor archives pay off when an engineer, or an agent acting for one, can find where a machine behaved like this before and read back exactly what it recorded. MorletDB makes that a database query. It keeps recordings bit for bit in PostgreSQL and turns bounded-lag search into thirteen shifted queries against one standard inner-product index, re-ranked with the unchanged scorer. Exact and probe search share one contract: hits are sample intervals of the stored recordings, filters and row-level security apply, and windows become searchable at the commit that builds them.

The measurements support three conclusions. First, a vector index helps only if it is asked the right question: a zero-lag shortlist loses a quarter of the top ten, whether computed exactly or approximately, while the probes recovered 97.7% on held-out seismic data at 57 ms instead of 740 ms (p95), and stayed 1.6–2.3× faster than a screened exact search with identical results on up to a million windows. Second, the costs are bounded and known: probe structures of 1.0–1.3× the raw data, exact search for small and selective galleries, and write-ahead logging from index maintenance (18× the database growth with default settings, 2.9× tuned). Third, shape is one signal among several: on public CNC milling data, bounded lag scored bad runs better than zero lag, but run length and level values did as well or better and the good history had to be recent, so a product should offer shape, level and context side by side. Time-resolved labels in a second signal domain are the next step for claims beyond seismic data.

Use of AI assistance and responsibility

Code, experiment harnesses, analyses and text of this report were developed with Claude (Anthropic) and reviewed and revised with Codex (OpenAI), under the authors' direction; Section 10 and the studies behind it were also written with AI assistance. Where a study had a protocol, it was committed before measurement (Section 5, the supplement and [57–59]); the authors reviewed the results and are responsible for the content and for any errors, each for the studies and text they contributed.

Licence

© 2026 Seung-Zin Nam. The text and figures of this report are licensed under Creative Commons Attribution 4.0 International (CC BY 4.0, https://creativecommons.org/licenses/by/4.0/); the MorletDB source code is licensed under Apache-2.0.

References

[1] pgvector contributors. pgvector: open-source vector similarity search for Postgres, version 0.8.0. https://github.com/pgvector/pgvector.

[2] Clara E. Yoon, Ossian O'Reilly, Karianne J. Bergen, and Gregory C. Beroza. 2015. Earthquake detection through computationally efficient similarity search. *Science Advances* 1, 11, e1501057. doi:10.1126/sciadv.1501057.

[3] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping. In *Proc. 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD '12)*, 262–270. doi:10.1145/2339530.2339576.

[4] GFZ German Research Centre for Geosciences and Institut des Sciences de l'Univers–Centre National de la Recherche CNRS-INSU. 2006. IPOC Seismic Network. Integrated Plate boundary Observatory Chile. Network code CX. doi:10.14470/PK615318.

[5] MorletDB contributors. 2026. MorletDB source repository, protocols and results. https://github.com/merkhet-io/morletdb.

[6] MorletDB contributors. R1b confirmation (2022). `evaluation/materialized/compact/results/conf-2022/RESULTS.md` in [5].

[7] MorletDB contributors. R2: retrieval scaling, and R2-D diagnostic. `evaluation/retrieval_scaling/` in [5].

[8] John Paparrizos and Luis Gravano. 2015. k-Shape: Efficient and Accurate Clustering of Time Series. In *Proc. 2015 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD '15)*. doi:10.1145/2723372.2737793.

[9] Abdullah Mueen et al. The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance. https://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.

[10] MorletDB contributors. BL1: shifted-probe retrieval, confirmation and ingestion. `evaluation/bounded_lag/` in [5].

[11] MorletDB contributors. SS1: one stored envelope representation. `evaluation/search_storage/SS1_RESULTS.md` in [5].

[12] MorletDB contributors. SS3: filtered envelope search. `evaluation/search_storage/SS3_RESULTS.md` in [5].

[13] MorletDB contributors. Pilot stages 1–2: access control, recovery and load. `evaluation/pilot/RESULTS.md` in [5].

[14] Davood Rafiei. 1999. On Similarity-Based Queries for Time Series Data. In *Proc. 15th Int. Conf. on Data Engineering (ICDE)*, 410–417. doi:10.1109/ICDE.1999.754957.

[15] Eamonn Keogh, Li Wei, Xiaopeng Xi, Sang-Hee Lee, and Michail Vlachos. 2006. LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures. In *Proc. 32nd Int. Conf. on Very Large Data Bases (VLDB)*, 882–893. Extended as: Eamonn Keogh, Li Wei, Xiaopeng Xi, Michail Vlachos, Sang-Hee Lee, and Pavlos Protopapas. 2009. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures. *The VLDB Journal* 18, 3, 611–630. doi:10.1007/s00778-008-0111-4.

[16] Michail Vlachos, Zografoula Vagena, Philip S. Yu, and Vassilis Athitsos. 2005. Rotation Invariant Indexing of Shapes and Line Drawings. In *Proc. 14th ACM Int. Conf. on Information and Knowledge Management (CIKM)*, 131–138. doi:10.1145/1099554.1099580.

[17] Leslie R. Foulds, Jorge P. de Morais Neto, Humberto J. Longo, Hugo A. D. do Nascimento, and Wellington S. Martins. 2015. A variant of

k-nearest neighbors search with cyclically permuted query points for rotation-invariant image processing. *Discrete Applied Mathematics* 197, 123–144. doi:10.1016/j.dam.2015.02.011.

[18] John Paparrizos and Luis Gravano. 2017. Fast and Accurate Time-Series Clustering. *ACM Trans. Database Syst.* 42, 2, 1–49. doi:10.1145/3044711.

[19] John Paparrizos and Michael J. Franklin. 2019. GRAIL: Efficient Time-Series Representation Learning. *Proc. VLDB Endow.* 12, 11, 1762–1777. doi:10.14778/3342263.3342648.

[20] Han Qiu, Hoang Thanh Lam, Francesco Fusco, and Mathieu Sinn. 2018. Learning Correlation Space for Time Series. arXiv:1802.03628.

[21] Kexin Rong, Clara E. Yoon, Karianne J. Bergen, Hashem Elezabi, Peter Bailis, Philip Levis, and Gregory C. Beroza. 2018. Locality-Sensitive Hashing for Earthquake Detection: A Case Study of Scaling Data-Driven Science. *Proc. VLDB Endow.* 11, 11, 1674–1687. doi:10.14778/3236187.3236214.

[22] Rigobert Tibi, Christopher Young, Antonio Gonzales, Sanford Ballard, and Andre Encarnacao. 2017. Rapid and Robust Cross-Correlation-Based Seismic Signal Identification Using an Approximate Nearest Neighbor Method. *Bull. Seismol. Soc. Am.* 107, 4, 1954–1968. doi:10.1785/0120170011. Read as the OSTI accepted manuscript (OSTI 1372350).

[23] Antonio Gonzales, Nicholas Blazier, et al. Seismic Waveform Matching via Approximate Nearest Neighbor with Projection Deficiency Compensation. Sandia National Laboratories, OSTI 1458306.

[24] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proc. 43rd Int. ACM SIGIR Conf.*, 39–48. doi:10.1145/3397271.3401075.

[25] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data. *Proc. VLDB Endow.* 13, 12, 3152–3165. doi:10.14778/3415478.3415541.

[26] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, et al. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proc. 2021 Int. Conf. on Management of Data (SIGMOD)*, 2614–2627. doi:10.1145/3448016.3457550.

[27] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proc. ACM Manag. Data* 2, 3, 1–27. doi:10.1145/3654923.

[28] Cheng Chen, Chenzhe Jin, Yunan Zhang, Sasha Podolsky, Chun Wu, et al. 2024. SingleStore-V: An Integrated Vector Database System in SingleStore. *Proc. VLDB Endow.* 17, 12, 3772–3785. doi:10.14778/3685800.3685805.

[29] Jiaye Wu, Peng Wang, Ningting Pan, Chen Wang, Wei Wang, and Jianmin Wang. 2019. KV-Match: A Subsequence Matching Approach Supporting Normalization and Time Warping. In *Proc. 35th IEEE Int. Conf. on Data Engineering (ICDE)*, 866–877. doi:10.1109/ICDE.2019.00082.

[30] Michele Linardi and Themis Palpanas. 2018. Scalable, Variable-Length Similarity Search in Data Series: The ULISSE Approach. *Proc. VLDB Endow.* 11, 13, 2236–2248. doi:10.14778/3275366.3275372.

[31] MorletDB contributors. E1: consolidated retrieval comparison, and diagnostics D1–D2. `evaluation/consolidated/RESULTS.md` in [5].

[32] Søren Kejser Jensen, Christian Schmidt Godiksen, Christian Thomsen, and Torben Bach Pedersen. 2025. Demonstration of ModelarDB: Model-Based Management of High-Frequency Time Series Across Edge, Cloud, and Client. *Proc. VLDB Endow.* 18(12), 5247–5250.

[33] Abduvoris Abduvakhobov, Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2024. Scalable Model-Based Management of Massive High Frequency Wind Turbine Data with ModelarDB. *Proc. VLDB Endow.* 17(13), 4723–4732.

[34] Magdalena Balazinska, YongChul Kwon, Nathan Kuchta, and Dennis Lee. 2007. Moirae: History-Enhanced Monitoring. In *3rd Biennial Conference on Innovative Data Systems Research (CIDR)*.

[35] YongChul Kwon, Guiping Xu, and Magdalena Balazinska. 2007. Identifying Similar Past Events in a Continuous Monitoring System. Technical Report UW-CSE-07-05-01, University of Washington.

[36] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. 2023. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 377–395.

[37] Jiayi Liu, Yunan Zhang, Chenzhe Jin, Aditya Gupta, Shige Liu, and Jianguo Wang. 2026. Fast Vector Search in PostgreSQL: A Decoupled Approach. In *Conf. on Innovative Data Systems Research (CIDR)*.

[38] Duo Lu, Helena Caminal, Manos Chatzakis, Yannis Papakonstantinou, Yannis Chronis, Vaibhav Jain, and Fatma Özcan. 2026. An In-Depth Study of Filter-Agnostic Vector Search on a PostgreSQL Database System. *Proc. ACM Manag. Data* 4, 3, Article 134. doi:10.1145/3802011.

[39] Abigale Kim, Marco Slot, David G. Andersen, and Andrew Pavlo. 2025. Anarchy in the Database: A Survey and Evaluation of Database Management System Extensibility. *Proc. VLDB Endow.* 18, 6, 1962–1976. doi:10.14778/3725688.3725719.

[40] Esteban Zimányi, Mariana Duarte, and Víctor Diví. 2024. MEOS: An Open Source Library for Mobility Data Management. In *Proc. 27th Int. Conf. on Extending Database Technology (EDBT)*, demonstration, 810–813. doi:10.48786/edbt.2024.76.

[41] Mahmoud Sakr, Esteban Zimányi, Alejandro Vaisman, and Mohamed Bakli. 2023. User-centered road network traffic analysis with MobilityDB. *Transactions in GIS* 27, 2, 323–346. doi:10.1111/tgis.12972.

[42] MorletDB contributors. Certified pruning for envelope search: headroom and feasibility. `evaluation/pruning/README.md` in [5].

[43] MorletDB contributors. Grid-spacing analysis on development data. `research/paper/analysis/` in [5].

[44] MorletDB contributors. E3: optimized exact baseline. `evaluation/strong_exact/RESULTS.md` in [5].

[45] MorletDB contributors. E6: a fast in-database exact scorer against probe search. `evaluation/fast_exact/RESULTS.md` in [5].

[46] MorletDB contributors. E7: screened exact envelope search. `evaluation/screened_exact/RESULTS.md` in [5].

[47] MorletDB contributors. E5 and E5-H: catalogued earthquakes, bounded-lag against zero-lag search. `evaluation/task_relevance/RESULTS.md`, `RESULTS_HELDOUT.md` in [5].

[48] MorletDB contributors. B3, B3.1–B3.12: MorletDB, QuestDB, CrateDB, Apache IoTDB, TDengine and TimescaleDB on dense waveforms; same-session pairs (B3.8, B3.9) and single-connection ingestion of the SQL 0.3.4 build (B3.10–B3.12). `evaluation/tsdb_b3/RESULTS.md` in [5].

[49] MorletDB contributors. B4: the chunk codec against the 2026 float codecs. `evaluation/storage_b4/RESULTS.md` in [5].

[50] MorletDB contributors. CS1 (S1 in the repository): per-chunk summaries. `evaluation/chunk_summaries/RESULTS.md` in [5].

[51] MorletDB contributors. EX1 (X1 in the repository): the full workflow inside edge-sized container limits. `evaluation/edge_x1/RESULTS.md` in [5].

[52] MorletDB contributors. EX2 (X2 in the repository): MorletDB on a Raspberry Pi 3B+. `evaluation/edge_x2/RESULTS.md` in [5].

[53] MorletDB contributors. B6: Grizzbit 2.1, chunk format version 5. `evaluation/storage_b6/RESULTS.md` in [5].

[54] MorletDB contributors. G1: automatic calibration-grid detection. `evaluation/grid_detection/RESULTS.md` in [5].

[55] MorletDB contributors. Read-only MCP server for AI agents. `tools/mcp/` and `docs/mcp.md` in [5].

[56] Mohamed-Ali Tnani, Michael Feil, and Klaus Diepold. 2022. Smart Data Collection System for Brownfield CNC Milling Machines: A New Benchmark Dataset for Data-Driven Machine Monitoring. *Procedia CIRP* 107, 131–136. doi:10.1016/j.procir.2022.04.022.

[57] Semir Maslo. 2026. IU1: bounded-lag retrieval on the Bosch CNC milling data, with addenda A1 (drift) and A2 (lead-in). `evaluation/industrial_iu1/` in [5].

[58] Semir Maslo. 2026. IU2: bounded-lag envelope distance and tool wear on milling tools run to failure. `evaluation/industrial_iu2/` in [5].

[59] Semir Maslo. 2026. IU1-E: search latency under edge-sized container limits (2 CPUs, 512 MB). `evaluation/industrial_latency/` in [5].

[60] Grzegorz Piecuch and Tomasz Żabiński. 2025. A new open dataset from a milling process – data for classification and estimation of tool life. *Scientific Data* 12, 650. doi:10.1038/s41597-025-04923-y. Data: Figshare, doi:10.6084/m9.figshare.28589216.

[61] MorletDB contributors. E4: scaling crossover between probe search and materialized exact search. `evaluation/scaling_crossover/RESULTS.md` in [5].

A Claims and evidence

Each claim with the study that supports it, the measurement kind (E: offline experiment control, P: product functions) and data (D: dev, C: held-out), and its main restriction. IU1 and IU2 are the industrial studies of Section 10 and IU1-E their latency under container limits; IU1-A1 (drift) and IU1-A2 (lead-in) are post hoc checks whose rules were written down before they were computed [57–59]. Protocols, results and commands: [5]; E1 and diagnostics: [31].

Claim	Study	Restriction
zero-lag top 1000 + re-rank: 0.743 (exact), 0.744 (HNSW)	R2-D; E, D	earlier build; two budgets
HNSW and exact shortlists share 98–100% of candidates	R2-D; E, D	largest gallery
zero-lag HNSW 1000 from probe index: 0.744 / 0.658	E1; P, D / C	one HNSW configuration
probes 0.977 (min 0.7; 5/100 <0.9), p95 57 ms	E1; P, C	seismic; unfiltered; quiet machine
fastest in-database exact p95 740 ms (8 workers allowed)	E1; P, C	PG 17 defaults, warm cache
materialized exact scan p95 21/42/80 ms vs probes 68/108/202 ms at 0.23/0.50/1.03 M windows	E3, E4; E, D	outside the database; no filters, RLS
faster exact path (identical lists) p95 277/575/1551 ms vs probes 55/68/238 ms	E6; P, D	8 workers; float64 identical
screened exact (identical lists) p95 138/221/540 ms vs every window 516/808/1928 ms vs probes 88/130/237 ms	E7; P, D	8 workers; one session, busier machine
probes 0.998, p95 53 vs 783 ms	E1; P, D	development data
exact grid candidates 0.999 (1 of 1,000 missed); float16 probes no further loss	E1; E, C	offline controls
HNSW candidates: 0.999 → 0.977 (held-out)	E1; P vs E	this configuration only
random 0.998, high-energy 0.956 (held-out)	E1; P, C	energy is a query rule, not relevance
83/12/4/1 held-out queries at 1.0/0.9/0.8/0.7	E1; P, C	3 of 23 misses were rank 1
same complete universe on both paths	E1 check	static databases
7-lag grid loses 0.013–0.014	E1; E, D+C	offline control
grid probe keeps ≥0.98 correlation (p5), rank ≤8 (p95)	E1; E, D+C	explains, does not guarantee
no short results; identical lists on the same index	E1; P, D+C	rebuilt index may differ

Claim	Study	Restriction
history score AUROC 0.925 (BL), 0.875 (ZL), 0.852 (RMS); BL vs RMS n.s.	IU1; E, C	run labels, all half-years as history
inside machine and half-year: BL 0.905 vs ZL 0.790; without M02 Feb 2019: 0.854	IU1-A1; E, C	post hoc, rules written first
earlier half-years only: BL 0.603 (n.s.), 13 bad runs	IU1-A1; E, C	half-year resolution only
without each run's first 2 s: BL 0.900	IU1-A2; E, C	query side only, 2 s assumed
run length alone: AUROC 0.982, better than BL	IU1-A1; E, C	run length from file length
other bad runs in top 10 inside the half-year: BL 0.254, ZL 0.212, RMS 0.377, chance 0.183	IU1-A1; E, C	replaces 0.177 vs 0.026 of IU1
same operation on another machine, post hoc rerun: BL 0.523, ZL 0.412, chance 0.076	IU1 T6; E, C	exploratory; original 0.503 withdrawn
stretch 1/2/5/10%: source in top 10 0.97/0.76/0.51/0.20	IU1; E, C	synthetic stretch
probes recall 1.0, p95 126 vs 176 ms at 85k windows	IU1; P, C	laptop, Docker
tool end of life AUROC: BL 0.639, ZL 0.580 (n.s.), RMS per window 0.791	IU2; E, C	7 tools; cycles to failure
vibration level 0.831, spindle current 0.813, cycle count 0.778	IU2; E, C	authors' feature table, not MorletDB
4.88 billion samples in 6.47 GB (10.6 bits per sample)	IU2; P, C	three accelerometers, 25 kHz
2 CPUs, 512 MB: history search p95 42 ms (43 unlimited), window statistics 1.6 ms	IU1-E; P, C	laptop cores, warm cache
2 CPUs, 512 MB: search over 85k windows p95 366 ms exact, 197 ms probes; no OOM	IU1-E; P, C	one client, two rounds

Claim	Study	Restriction
index statements ≈65% of probe time; 0.7 ms per probe unresolved	D1; P, D	instrumented copy
correlated filters: recall 0.997–1.0, p95 121–227 ms	D2; P, D	synthetic filter choice

Claim	Study	Restriction
post-filter 0.515 at 1.8% eligible; routed 1.0	SS3; E/P, C	earlier build
routed recall 0.977–1.0, no filter or RLS violations	SS3; P, D+C	earlier build
probe structures 0.97–1.34× raw; rebuild 39–41 s	E1; P	depends on raw compressibility
one stored copy: −250 MB, +23% or +27–32% p95	SS1; E, D+C	earlier build
lag-uniform bounds prune ≤63% (idealized)	pruning; E	one construction; model
searchable at commit; 919k samples/s	BL1-S; E, D	one day, four channels
409.6k samples/s: 15 min (defaults), 111 min (tuned)	C2, C4; P	2-h run not completed
WAL/growth 18× (defaults), 2.9× (second point)	C2, C2p; P	four settings and load differed
companion-table RLS: exact +5%	Stage 1; P	one tenant layout
calibrated floats: codec 0.62–0.76× ClickHouse, TileDB	B2; E	six inputs; ClickHouse 26.5
smallest on 4 of 6 waveforms; Pcodec 5–8% smaller on 2; ALP 2.6–7.9× larger on calibrated	B4; E	default settings; one machine
grid detected without metadata: 0.98–1.00× declared-quantum size	G1; E	six inputs
Grizzbit 2.1: waveforms 0.874× in total, smaller on 6 of 6; decode ≥53.1, encode ≥10.8 M/s	B6; P	historical B6 build

Claim	Study	Restriction
sample data 33% / 87% smaller than QuestDB Parquet zstd 9 / native	B3; P	one dataset; needs a value grid
released MorletDB: QuestDB ingests 1.5× / 1.4× faster (one connection); MorletDB reads 4,096 samples 1.6× / 2.3× faster; long reads, statistics no statement	B3.8; P	one machine; 8-CPU containers; same session
MorletDB ingests 1.2–2.0× faster with 4–8 parallel writers	B3.2; P	one machine; 8-CPU containers
historical SQL 0.3.4 build: ingests 1.4× / 1.6× faster than QuestDB (one connection); data identical, reads unchanged	B3.10, B3.12; P	paired with QuestDB only; one dataset, one machine
CrateDB 6.4: 10.8× space, ingest about 8.5× and reads 6.4–12× slower; altered resend ignored	B3.4, B3.9; P	one connection; single node
IoTDB 2.0: 4.0× space, ingest about 2.3–2.6× and reads 3.1–10.7× slower; altered resend replaces	B3.5, B3.9; P	one connection; Python client decoding included
TDengine 3.4: 3.4× space, ingest about 1.2× and reads 3.6–6.8× slower; tsz lossy	B3.6, B3.9; P	one connection; Python client decoding included
TimescaleDB 2.31: 4.9× space (columnstore), ingest about 6.5× and reads 2.1–5.4× slower; every resend refused	B3.7, B3.9; P	one connection; psycopg row conversion included

Claim	Study	Restriction
search features at ingest: 3.9× raw ingest time (was 9.6×)	B3.3; P	one connection, one dataset
seven-channel acquisition reads 32–39% faster than QuestDB (p50)	B3.3; P	one dataset
summaries: statistics 58× faster (p95, 60 s at 64 kHz), bitwise identical; ingest −4.5%, storage +1.3%	CS1; P	gain grows with interval length
full workflow at 2 CPUs, 512 MB, no slowdown	EX1; P	container limits on desktop cores
Raspberry Pi 3B+: 78k / 258k samples/s, search lists identical	EX2; P	one device, no fan; not long-term
bounded lag P@10 0.297 vs 0.213 (dev), 0.139 vs 0.060 (held-out)	E5, E5-H; E, D / C	catalogue proxy; exact search