Supplement to "MorletDB: Recordings, Not Rows"

Version 0.31

Seung-Zin Nam
Germany

Semir Maslo
Sulzer-BDiM
Germany

About this supplement

This supplement holds studies that support the system described in the main paper but are not needed for its argument: lossless storage of raw samples, comparisons with the 2026 float codecs and with a time-series database, interval statistics from chunk summaries, envelope precision, building representations outside or inside the database, access control, recovery and load tests, small machines, agent access, and application evidence and negative results. Text, tables and numbers are carried over from draft 0.6 of the paper; protocols and raw results are in the repository [15]. Section and table references to "the paper" refer to the main manuscript.

The machine, PostgreSQL and pgvector versions are those of the main paper. In the storage study P2, TimescaleDB 2.30.2 ran in Docker and we compare only its sizes; B3.7 and B3.9 compare TimescaleDB 2.31 on storage and speed (Section 2.4).

1 Lossless chunk coding

Each raw chunk is one `bytea` value, coded by Grizzbit, MorletDB's lossless chunk codec (Grizzbit 1 is chunk format versions 2 and 3). The codec is a named, specified format, not a new algorithm; it combines established techniques: fixed polynomial predictors of order 0–4 as in Shorten and FLAC [2, 3], partitioned Rice coding of zigzag mapped residuals [4], and a CRC-32C checksum. Its distinctive part is how it treats floats.

Most sensor values are calibrated integers: a digitizer emits counts d, and acquisition software stores $x = f(d)$, for example $x = d \cdot s$ (seismic counts, WAV) or, in EDF [5], $x = (d - d_{\min})(p_{\max} - p_{\min})/(d_{\max} - d_{\min}) + p_{\min}$. The recording declares a nominal quantum q: 1 for counts, 0.001 for three decimals, the ADC step for calibrated values. Per chunk, the encoder seeks a grid

$$x_i = o + k_i\, s \oplus \delta_i, \tag{1}$$

with integer indices k_i, a step s refined from q to within one unit in the last place (ulp), an offset o (format version 3; zero in version 2), and corrections δ_i, stored as differences of order-preserving integer images of float64, so that decoding reproduces every bit pattern. A division mode $x = k/d$ covers decimal steps exactly. Indices are predicted and Rice-coded like integers; corrections form a second stream. If no grid fits within a thousandth of a step, the chunk is coded as order-preserving float bits, and if that is not smaller, as raw float64. The quantum is a storage hint, never a conversion: a wrong quantum costs space, not accuracy.

Detecting the grid (G1). Floats sent without a declared quantum are now examined chunk by chunk for a value grid: the encoder estimates a candidate step from the values themselves, refines it as above, and keeps it only if the grid coding is smaller than float coding; otherwise the chunk is coded as before. On B1's six inputs with quantum 1, detection matched the declared-quantum size (Table 1), was never more than 0.1% larger than the previous encoder, and round-tripped every chunk bit for bit; on the four float inputs it was 4.1–7.8× smaller than the previous encoder and encoded faster than its float coding (ECG 11.1 against 3.3 M samples/s) [30]. Floats without any value grid still fall back to float coding.

Table 1: G1: bits per sample with a declared quantum (D), with quantum 1 and detection (A), and with quantum 1 and the previous encoder (P).

Input	D	**A**	P	A/D
CX counts	4.61	**4.61**	4.61	1.00
IMS 3-decimal	8.81	**8.81**	63.02	1.00
Paderborn counts	7.88	**7.88**	7.88	1.00
Paderborn physical	9.17	**9.17**	63.17	1.00
Sleep-EDF EEG	14.97	**14.66**	60.02	0.98
PTB-XL ECG	7.40	**7.40**	57.58	1.00

Table 2: P2: total database storage in bits per sample. T: the TimescaleDB layouts tested (a row per sample with an integer or float column; a row per 4096-sample array).

Input	MorletDB	T int	T float	T array
CX seismic counts	**6.12**	8.25	14.34	25.0
IMS three-decimal	**11.36**	14.06	71.95	28.1
Paderborn counts	**9.95**	12.89	24.58	26.4
Paderborn physical	**11.53**	—	70.12	29.5

2 Storage of raw samples

2.1 Database footprint (P2)

P2 appended the same 4,194,304 samples of four inputs (CX seismic counts [11], IMS bearing vibration [14], Paderborn bearing vibration [13] as ADC counts and as physical floats) to MorletDB and to TimescaleDB hypertables with native compression (Gorilla for floats; delta-of-delta and Simple-8b for integers), and measured total on-disk bytes, including indexes, TOAST and metadata tables, after verifying that four intervals per layout read back bit-exactly [16].

TimescaleDB on integer columns comes closest but is 24–35% larger (Table 2), and requires the user to convert values to counts first. On float columns, Gorilla's XOR coding *expands* noisy decimal and calibrated values beyond 64 bits per sample. We tested one `segmentby`/`orderby` configuration and not TimescaleDB's newer column formats. Of MorletDB's footprint, the codec payload is 4.6–9.2 bits per sample (Table 3); PostgreSQL row and TOAST overhead adds 1.5–2.6 bits per sample at 4096-sample chunks.

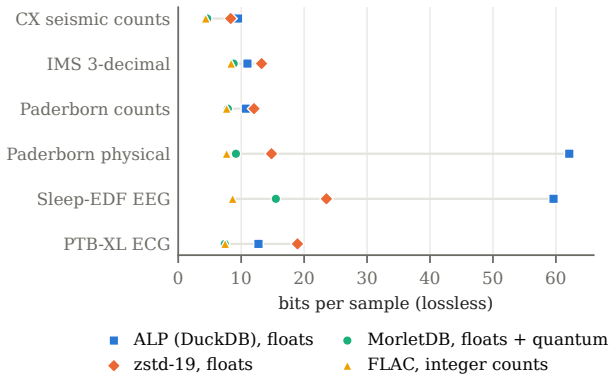


Figure 1: B1: lossless bits per sample on identical samples. Float codecs see only float64 values; MorletDB also sees the nominal quantum; FLAC sees integer counts. Table 3 gives all arms.

Table 3: B1: bits per sample. Z: zstd-19; PQ: Parquet; DDB: DuckDB.

	float input				integer input		
Input	Z	PQ	ALP	**MorletDB**	FLAC	PQ	DDB
CX counts	8.33	6.84	9.50	**4.61**	4.38	5.24	6.38
IMS 3-decimal	13.24	42.94	11.00	**8.81**	8.41	9.87	10.88
Paderborn counts	12.04	9.48	10.75	**7.88**	7.70	9.61	11.25
Paderborn physical	14.82	30.57	62.13	**9.17**	7.70	9.61	11.25
Sleep-EDF EEG	23.54	38.15	59.63	**15.51**	8.65	10.22	12.00
PTB-XL ECG	18.94	44.64	12.75	**7.40**	7.46	10.27	12.75

2.2 Codec against modern baselines (B1)

B1 compares codecs on identical samples, each verified lossless [17]. Float codecs receive only float64 values: zstd level 19 per chunk, Parquet with `BYTE_STREAM_SPLIT` and zstd, and DuckDB 2.0's automatic float compression, which uses ALP and its fallback ALP-RD [1, 7]. MorletDB receives the values plus the nominal quantum. Integer codecs receive the exact digitizer counts: FLAC [3], Parquet delta coding with zstd, and DuckDB bitpacking. Two inputs extend the earlier storage inputs: a Sleep-EDF EEG slice [8, 9] from 32 nights with symmetric and asymmetric calibration ranges (an earlier engineering check had measured MorletDB's own storage of this collection, not the other arms), and PTB-XL ECG lead I [10], new to the storage studies. The Parquet float arm is one fixed encoding, not a tuned Parquet baseline; ALP is DuckDB 2.0.0's implementation, measured as table blocks.

Figure 1 and Table 3 show three regimes.

Counts and decimals. When values are integers or short decimals (CX, IMS, Paderborn counts, ECG with gain 1000), ALP and MorletDB both recover integers; MorletDB is 20–51% smaller than ALP, plausibly because consecutive samples are correlated and prediction exploits that, while ALP's frame-of-reference bit-packing does not; B1 does not isolate this. MorletDB is within 5% of FLAC, whose LPC stage it lacks, and slightly below FLAC on ECG.

Calibrated floats. On Paderborn physical values and Sleep-EDF, no decimal exponent turns values into integers, and the tested DuckDB ALP implementation selected ALP-RD, storing about 60 bits per sample, near raw. Of the three metadata-free arms, zstd was smallest. MorletDB, which additionally receives the nominal quantum, was 1.5–1.6× smaller than zstd. Our pre-registered claim of at least 2× was not met. B2 added ClickHouse (26.5, best of four float codec chains) and TileDB (2.30, best of four filter pipelines) on the same samples [23]: on the calibrated inputs MorletDB was 0.66 and 0.76× the best of them (ClickHouse `ZSTD(19)` on both), so it remained the smallest metadata-free arm; FLAC remained the best integer codec on every input.

The correction gap. Given counts, MorletDB codes the two calibrated inputs at 7.9 and 8.8 bits per sample; given the calibrated floats, it needs 1.3 and 6.7 bits per sample more. That extra cost, relative to MorletDB's own integer-input path, is the correction stream: the source software computed x with a particular sequence of roundings (on Sleep-EDF a multiplication, a division and an addition), which a single fitted step and offset does not reproduce for every sample. It does not explain the whole difference from FLAC, which also includes MorletDB's integer path being 0.15–0.18 bits per sample larger than FLAC's. Replaying the source's calibration arithmetic might remove the corrections when the correct parameters and operation sequence are known; this is neither implemented nor tested. Our pre-registered claim that MorletDB stays within 15% of the best integer codec held on four of six inputs and failed on these two.

For float compression, these inputs illustrate that real-valued sensor data can be neither decimal nor arbitrary but an affine image of integers under floating-point rounding. On the two such inputs we tested, the decimal-oriented ALP implementation fell back to near-raw size and the best byte-oriented arm was 1.5–1.6× larger than coding with a declared quantum.

2.3 Against the 2026 float codecs (B4)

B4 repeated the comparison on B1's six waveform inputs and 22 decimal series of the Elf suite against codecs released or extended since: Pcodec 1.0.4, Parquet's ALP encoding (Rust `parquet` 60.0.0) and ClickHouse 26.10 with ALP and PCO, all at default settings, every arm decoded and compared bit for bit [26]. Table 4 shows the waveforms. MorletDB was the smallest on four of six. Pcodec was smaller on IMS decimals (8%) and Paderborn physical values (5%), and ClickHouse's PCO on the latter (3%): these inputs are almost, but not everywhere, multiples of one step, which Pcodec's float-multiple mode with per-chunk adaptation captures and MorletDB's single grid per chunk does not. On calibrated values every ALP arm was 2.6–7.9× larger than MorletDB; ALP's decimal detection does not find a calibrated ADC grid. On the Elf series MorletDB was the smallest on 12 of 22 and Pcodec on 9; five losses were large (2.7–4.1× Pcodec) on high-precision series without a grid. The storage lead on waveforms therefore holds against databases, but not against every codec on every input.

2.4 Against a time-series database (B3)

B3 compared MorletDB with QuestDB 10.1.0 (open source) on 124 Paderborn acquisitions [13]: about 4 s each of three 64 kHz

Table 4: B4: lossless bits per sample on waveforms, default settings (smallest in bold). ALP: the smallest of the Parquet and ClickHouse ALP arms, with or without zstd.

Input	MorletDB	Pcodec	CH PCO	ALP
CX counts	**4.61**	5.23	5.33	9.26
IMS decimals	8.81	**8.12**	9.19	10.13
Paderborn counts	**7.88**	8.31	8.47	10.36
Paderborn physical	9.17	**8.68**	8.92	24.02
Sleep-EDF	**14.66**	20.27	18.33	40.81
ECG	**7.40**	7.84	8.08	10.87

Table 5: B3.8: the released MorletDB (0.3.5-alpha.1) against QuestDB 10.1, loaded alternately in one session, one client connection unless stated; medians of three loads; p50 / p95 where two numbers are given. Bold: better by at least 10% (in all loads, or at both p50 and p95).

Measure	MorletDB	QuestDB
sample data on disk (MiB)[b]	**156.8**	232.7 / 1,162.7[a]
ingestion, asynchronous, M samples/s	6.30	**9.66**
ingestion, durable, M samples/s	6.15	**8.48**
server memory, median peak (MiB)	**168**	2,301
read 4,096 samples (ms)	**0.63 / 1.01**	1.02 / 2.29
read 128,000 samples (ms)	6.89 / 9.31	6.01 / 10.16
read 3 × 64,000 samples (ms)	**7.38 / 10.28**	8.75 / 12.92
4 readers, queries/s	**4,192**	2,647
min, max, mean, RMS (ms)	2.54 / 2.69	2.46 / 4.73
ingestion, 8 writers, M samples/s (B3.2)	**16.47**	10.10
SQL 0.3.4 build (I1, B3.10), asynchronous / durable, M samples/s	**13.65 / 12.99**	9.58 / 8.26

[a]Parquet zstd 9 / native columns. [b]From B3.2; QuestDB's files and MorletDB's database had the same size in B3.8.

channels (vibration, two motor currents), three 4 kHz channels and one 1 Hz temperature, 101,586,735 samples or 775 MiB as float64 [31]. Each database ran alone in a container limited to 8 CPUs and 16 GB, loaded the samples through its recommended Python client path, and passed every correctness check after every load: each of 868 channels per load read back with the manifest's SHA-256, exact timestamps, and random intervals crossing ingestion batches. Statements need at least 10% in all loads, or at both p50 and p95. Stage 1 was frozen before any measurement; two amendments (B3.1, B3.2) changed MorletDB's client path and server functions, were tuned on a disjoint development set, and were confirmed once on the benchmark set with QuestDB rerun in the same session. Later stages (B3.4–B3.7) compared other systems with MorletDB's B3.3 numbers from another session; B3.8 and B3.9 repaired that by loading each system alternately with the released MorletDB (0.3.5-alpha.1, SQL 0.3.3) in one session on a quiet machine, and their ratios replace the earlier speed statements. The server changes of B3.1 and B3.2 are SQL 0.3.3 functions that append and read samples as raw little-endian float64 bytes and compute interval statistics while decoding, plus pipelined appends, zero-copy array arguments and a faster decoder (chunks byte-identical).

Table 5 shows the confirmed state. MorletDB stored the sample data in 33% less space than QuestDB's smallest arm (Parquet zstd 9, an explicit conversion step that never covers the newest partition) and 87% less than its native columns; its whole database (164.8 MiB) was 29% smaller than that arm's data. From one client connection QuestDB ingested about 1.5× faster with asynchronous and 1.4× with durable commits (every QuestDB load at least 1.52× and 1.36× every MorletDB load; B3.2 had found 1.3–1.6× and 1.2–1.4× against an earlier build), with 14× MorletDB's server memory. With four or eight parallel writers per system (B3.2, same session) MorletDB ingested 1.2–2.0× faster, below four of its eight cores, while QuestDB used most of them. Reads of 4,096 samples were 1.6× / 2.3× faster in MorletDB and three-channel reads 1.2× / 1.3×; long reads (QuestDB 13% faster at p50, MorletDB 9% at p95) and interval statistics (on par at p50, MorletDB 1.8× at p95) gave no statement, and four concurrent readers achieved 1.6× the queries per second. A repeat at the end of the night reproduced the ingestion figures within 1% and the same read statements. A retry with one altered sample was refused by MorletDB and silently replaced the stored value in QuestDB (deduplicating upsert). Stage 1, before the amendments, had QuestDB ingesting 13–15× faster and reading long intervals 2–3× faster; the gap was MorletDB's client path.

Next release (I1). The importer and the append path were then changed (issue #233: importer pacing, batched raw appends, threaded encoding; SQL 0.3.4, historical measured build), with development on the development set only [15]. B3.10, frozen before its first benchmark load, loaded it alternately with QuestDB 10.1.0 and the released MorletDB in one session, three loads per durability class. All 18 loads passed every correctness check, and the stored chunks were byte-identical to the released build's in every load. From one client connection it ingested 13.65 and 12.99 M samples/s (asynchronous / durable) against QuestDB's 9.58 and 8.26 M: 1.4× and 1.6× faster (every load at least 1.40× and 1.50× every QuestDB load), and 2.2× / 2.1× the released build, at 177 against 2,307 MiB peak server memory. The session reproduced B3.8 for the released build (QuestDB at least 1.47× / 1.16× faster). Its read criterion, every query's p50 at most 1.1× the released build's, failed in B3.10 (128,000 samples, 1.12×) and in B3.11 over three loads per arm (three-channel reads 1.27×, four readers 1.23×). An exploratory investigation found no read defect in the database: the server's own read cost for data written by I1 was within 4% of that for data written by the released path, and more than half of a measured read was client, container network path and conversion. B3.12, frozen with reads from a fresh client process and server-side read cost, met every criterion (client p50 0.99–1.05×, server-side 1.00–1.04×), so the change was adopted. This is one dataset on one machine with 8-CPU containers, paired with QuestDB only; the other systems below are compared with the released build.

Three conditions bound these results. The storage advantage comes mostly from coding values on their grid: without it (an attribution run outside the frozen arms) MorletDB's sample data grew to 621.6 MiB, larger than QuestDB's Parquet arm; with grid detection (G1) and no quantum declared it was 151.1 MiB, 35% smaller than that arm. On the whole volume, PostgreSQL's retained write-ahead log made QuestDB's Parquet arm 33% smaller with default WAL settings; with a smaller WAL limit the two were within 3% (240.6 against 247.3 MiB). And this is one dataset on one machine,

a map of trade-offs rather than a ranking. Similarity search was not part of the comparison.

A third amendment (B3.3) measured two things QuestDB does not offer in the same form. Reading all seven channels of one acquisition over a time interval with one server call took 1.76 / 2.37 ms (64 ms of data) and 5.76 / 6.88 ms (0.5 s) at p50 / p95, 32–39% faster than QuestDB at p50 with one query per group table; every sample, first index and time was exact. Computing search features during ingestion, with a faster feature extractor adopted under a frozen rule (features bitwise identical to the previous one), reduced ingestion from 6.33 to 1.64 M samples/s from one connection, 3.9× the ingest time instead of 9.6×; features added 130 MB and envelopes 53 MB to a 165 MiB database. The extractor remained the largest single cost (49 s of server time in an 87 s profiled load).

Three further amendments added databases aimed at industrial IoT, and a fourth TimescaleDB, each frozen before its first benchmark load. They compared with MorletDB's B3.3 numbers from another session; B3.9 re-paired each system's default or fastest layout with the released MorletDB in one session (60 loads, all correct), and its ratios replace their speed statements, while their storage, correctness and semantics stand (storage was re-measured identical; Table 6). CrateDB 6.4 (B3.4) ran as a single node through its Python client over HTTP in three layouts: a row per sample instant, the same with its best compression, and arrays of 4,096 samples per row, its documented pattern for dense series. Its timestamps have millisecond resolution, so every layout stored the sample index. One load overlapped a sleep of the machine and was rerun from a fresh volume under the stage's rule. Apache IoTDB 2.0.11 (B3.5) ran standalone in its table model with nanosecond timestamps, so times were exact without an index, in two layouts: its defaults (GORILLA, LZ4) and ZSTD. Its Python client converts a timestamp column element by element, ten times slower than reading the time as an integer, which the protocol therefore did; part of its remaining read time is client-side decoding (the server answered a 128,000-sample query in about 20 ms in development). TDengine 3.4.3.3 (B3.6) ran as a single node with nanosecond timestamps through its Python WebSocket client, a subtable per acquisition under supertables, in three layouts: its default (byte-stream split, LZ4), ZSTD at the highest level, and its float compressor tsz. The first two returned every sample exactly in all 12 loads; tsz did not: two channels differed in 54 samples by at most 8 ULP, and TDengine's server reference documents a mantissa truncation of doubles below 10^{-16} (default and minimum) for it, so it is lossy and was stopped after its first load. All 37 lossless loads returned every sample exactly. TimescaleDB 2.31.0 (B3.7) ran on PostgreSQL 17 with default settings, an integer nanosecond time dimension, a unique index and binary COPY through psycopg, measured in its rowstore after ingestion and in its columnstore after compression; all 6 loads were exact in both states, and its columnstore came out at about the raw float64 size. With the same-session ratios, MorletDB ingested about 8.5× faster than CrateDB's array layout (18× its row layout), 2.3–2.6× faster than IoTDB, 1.2× faster than TDengine and 6.5× faster than TimescaleDB, and read intervals 6.4–12×, 3.1–10.7×, 3.6–6.8× and 2.1–5.4× faster. The earlier cross-session ingestion figures (about 15×, 2.5×, 1.5× and 8.5×) overstated the gap: those stages most likely ran on a busy machine, which slows client-heavy paths most. IoTDB's default layout stored the data at about the raw float64 size; XOR-style coding gains little on noisy waveforms. Neither system offers similarity search over an archive; IoTDB's optional function library computes DTW or cross-correlation between two given series.

Table 6: B3.4–B3.9 against databases for industrial IoT and TimescaleDB. Data on disk: each system's most compact lossless layout (B3.4–B3.7) as a multiple of MorletDB's 165 MiB database; for TDengine its data files (its whole volume, with 1 GiB of retained WAL, was 1,602 MiB). All other rows: B3.9's same-session ratio to the released MorletDB, each system's median divided by MorletDB's (> 1: MorletDB better); p50 / p95 where two numbers are given. CrateDB: the better of its array and row layouts per measure (^{r}: rows); IoTDB and TDengine: default layouts; TimescaleDB: columnstore. MorletDB refuses an altered resend.

Measure	CrateDB	IoTDB	TDengine	Timescale
data on disk	10.8	4.0	3.4	4.9
ingestion, async. / durable	8.6 / 8.4	2.3 / 2.6	1.18 / 1.21	6.5 / 6.4
read 4,096 samples	8.0 / 7.5	10.7 / 9.2	6.0 / 5.8	2.4 / 2.1
read 128,000 samples	7.1 / 6.4	6.0 / 4.9	6.8 / 5.6	5.4 / 4.5
read 3 × 64,000 samples	12.2 / 9.3^r	3.8 / 3.1	4.2 / 3.6	4.0 / 3.4
min, max, mean, RMS	1.4 / 2.9^r	2.6 / 3.2	1.5 / 3.9	4.1 / 4.2
4 readers, queries/s	5.2	6.7	14.9	3.8
server memory, peak	36.5	35.9	6.1	2.4
altered resend	ignored	replaces	replaces	refused[u]

[u]By its unique index, which also refuses an identical resend.

3 Interval statistics from chunk summaries (CS1)

Interval statistics decode every chunk of an interval, so their time grows with its length. With summaries enabled, each chunk also stores count, minimum, maximum and compensated sums in 66 bytes; statistics and overviews of an interval decode only the two chunks its edges cut and merge the stored summaries in order, so results are bitwise those of decoding everything [27]. The study (S1 in the repository, frozen before its run) used one hour of 64 kHz Paderborn vibration (230.4 M samples, built from the B3 development set) and one day of 100 Hz seismic data from the development gallery. All 3,000 statistics and overviews were bitwise identical with and without summaries; beyond the previous limit of 16,777,216 samples, results matched numpy within 6.4×10^{-15} relative. At p95, 60 s at 64 kHz took 1.7 instead of 100 ms (58×; frozen threshold 10×), 16.8 M samples 3.0–3.3 instead of 418–429 ms, and an hour 36 ms; at 100 Hz a day took 3.4 instead of 174 ms. One-chunk intervals gain little (2.7× at 1 s). Ingestion was 4.5% slower and the chunk table 1.3% larger; chunks stored before enabling are summarized at about 5,000–6,000 chunks/s.

Table 7: R1b on 2022 windows. Size relative to the raw chunk table; latency includes the client path.

Envelopes	Size/raw	Recall@10	Same order	p95 ms
from raw (reference)	—	1	1	1324
float32	28.1%	1.000	1.000	14
float16	**12.9%**	**0.9999**	**0.987**	**16**
uint8	7.2%	0.998	0.845	17
uint8, coarser	4.2%	0.952	0.067	16

4 Envelope precision and zero-lag storage (R1b, P1)

4.1 Precision of materialized envelopes (R1b)

R1b selected an envelope precision on 2023 seismic windows and evaluated it once on 2022 event and noise windows that, according to its report, no earlier study had read: 1,826 windows at ten CX stations [11], 1,557 event windows from 284 catalogued events and 269 no-match distractors [18]. The reference decodes raw samples and computes envelopes in the client.

Float16 envelopes met all frozen criteria (Table 7): recall@10 of 0.9999, 12.9% of the raw table, and an unchanged catalogue-distance relevance proxy (+0.001 km, 95% interval $[-0.002, +0.004]$). The roughly 80-fold latency reduction measures materialization against a client that decodes and recomputes; it is not a speedup over an optimized raw-scanning engine.

4.2 Same storage, different query (P1)

P1 stored the identical float16 envelopes in a pgvector `halfvec(410)` column [6]. Both tables occupied exactly 1,794,048 bytes, and all 1,826 envelopes were bit-identical, so envelope storage itself is no advantage. Zero-lag cosine search, however, shared only 64.7% of the lag-tolerant top ten and reproduced its order for 0.1% of queries. Scoring lags in the client on the same `halfvec` rows reproduced MorletDB's ranking exactly. In a later same-run comparison, median latency was 6.85 ms for MorletDB, 0.49 ms for zero-lag cosine and 20.3 ms for client lag scoring. The first two answer different questions, so their ratio is not a like-for-like speed comparison.

Representation overhead (R2 [19]). The materialized envelopes took 12.9% of the raw chunk table in R1b but 60.2% in R2. Measured sizes account for the factor of 4.68: halving the stride from 4096 (one envelope per event window) to 2048 doubles the number of envelopes per sample (×2.0); R2's continuous, mostly ambient recordings compress to 5.04 payload bits per sample against 12.39 for R1b's event windows (×2.46); row and TOAST overheads differ slightly (×0.95 combined). The ratio thus depends on stride, raw compressibility and physical layout, and the 12.9% figure does not transfer to continuous archives. R2's benchmark held four structures side by side: raw chunks (371 MB, always required), envelopes (223 MB, used by the exhaustive scan and every re-rank), the `halfvec` copy (241 MB) and its HNSW index (275 MB). The copy stores the same values as the envelopes, so a deployment would keep one of the two if its scorer read that format (MorletDB's reads its own); the index is needed only for an index-based shortlist.

5 Building representations (CW1)

As a first step toward compaction-style workers that run representation work outside the database, CW1 compared building envelopes inside PostgreSQL with stateless external processes that read encoded chunks through SQL, compute with the same Rust code, and write rows back with key-idempotent inserts [25]. On 33,736 windows the external path produced byte-identical rows in every run, survived a killed and restarted worker and two workers processing the same recordings, and moved 90% of the per-window CPU off the database server (0.13 against 1.31 ms of database CPU per window). It also used about four times less CPU in total, which exposed the real inefficiency: the in-database builder issued one SQL statement per window and decoded every raw sample twice. Batching the in-database builder, so that each chunk is decoded once for up to 256 windows, reduced database CPU to 0.31 ms per window and raised one session from 740 to 3,084 windows per second (four sessions: 12,034), byte-identical to before. External workers remain a viable basis for separating compute from storage, but on one machine batching delivered most of their benefit without them.

6 Access control, recovery and load

Before a bounded pilot with human review of results, we tested access control, recovery and sustained load against plans frozen before each run [24]. All runs shared the machine with an unrelated CPU-intensive job (load average 3–20), so latencies are indicative; correctness results are not affected.

Access control. With row-level security only on the recordings table, a tenant could still read other tenants' chunks, envelopes and probe vectors by querying the companion tables directly: search functions honoured the policy, the tables did not. `morlet\_enable\_row\_security` now applies a policy to every companion table that admits rows whose recording is visible. Its first form, a correlated `EXISTS`, prevented parallel plans and tripled exact search time (946 to 2,821 ms on 235,000 windows); comparing against an array of visible recording identifiers kept the parallel scan (995 ms, +5%) and cost probe search 6 ms. We did not test policies with thousands of visible recordings. Acquisitions, added later, had a gap that this test did not cover: in SQL 0.3.1 and 0.3.2 only a foreign key checked a channel row's acquisition, so a role allowed to add 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, through the function and through direct `INSERT` and `UPDATE`, only into acquisitions the caller can see; a new acquisition is visible to its creating transaction until its first channel is added. A two-tenant regression test with 16 checks runs in continuous integration (advisory GHSA-pg6f-hcvw-vw34).

Recovery. On a 50,604-window database every check passed: each recording hashed identically afterwards, each complete window had exactly one envelope and probe row, and exact searches returned identical lists. A full dump restored in 18 s; dumping only recordings and chunks took a third of the space (146 against 459 MB), and rebuilding envelopes and the probe index after restore took 23 s. Twenty backends terminated at random points inside append transactions left no partial rows, and replaying the chunk succeeded; aborted builds resumed, and an aborted probe

enable could be retried. Killed and duplicated external workers produced exactly one byte-identical row per window, but did not write probe rows, so their windows stayed invisible to probe search until filled. After deleting 15% of recordings, `VACUUM` took 113 s and left the index size unchanged, whereas an online `REINDEX CONCURRENTLY` took 10 s while four clients searched (p95 23 to 31 ms). An extension update forced to fail midway rolled back completely.

Replay. Replaying one day of four development channels (16,868 windows) with the builder called in each append transaction, every window was searchable at its commit while appends ran at 919,000 samples/s, 53% of raw-only throughput. At 200× real time, probe queries during ingestion had a p95 of 82 ms against 189 ms for exact search, at recall 0.9997 on the same snapshot [22].

Sustained load. On an archive of 938,768 windows (4.8 GB), sixteen channels at 25.6 kHz (409,600 samples/s) were appended by four writers, each building envelopes and probe rows in the append transaction, while four analysts issued one query every two seconds, half filtered to about 2% of windows and half unfiltered. With PostgreSQL's default configuration the run met its criteria (filtered p95 167 ms; probe p95 337 ms, spot-check recall 0.998; ingest lag at most 22 s, recovered), but wrote 13.8 GB of write-ahead log for 0.76 GB of database growth in 15 minutes: HNSW insertions touch pages across a 1.2 GB index, and each checkpoint forces full-page images of them. At twice the rate the writers fell behind without recovering; capacity was about 1.3–1.4× the target. With a pilot configuration (8 GB shared buffers, 16 GB `max\_wal\_size`, 15-minute checkpoints, WAL compression, changed together) the same run wrote 1.91 GB of WAL for 0.65 GB of growth, kept ingest lag at or below 2 s, and gave filtered and probe p95 of 81 and 100 ms (recall 0.994); background load was lower in this run (7.3 against 11.7), which confounds the latency comparison but not the WAL volume. Unfiltered exact search over the full archive stayed at 7.8–9.7 s p95, which is why the router sends large galleries to probes. A run planned for two hours with the same configuration met the target for 111 minutes (409,585 samples/s; writer lag at most 24 s in the first hour, with the unrelated job running, and 3.4 s in the second; filtered and probe p95 118 and 138 ms, recall 0.993; no errors; 12.0 GB of WAL for 4.7 GB of growth). Then all client processes paused for 90–106 s; the server log shows no error, which does not rule out a system-side cause; and the harness's lag rule, stricter than the protocol's wording, stopped the run at 112.5 minutes; the stored result divides the ingest rate by the scheduled duration, and a re-analysis from the raw logs reports the rate over the elapsed time. The two-hour run therefore did not complete. A measured stability duration is not a production qualification.

7 Small machines (EX1, EX2)

Two frozen studies (X1 and X2 in the repository) ran the full workflow on B3's benchmark set (124 acquisitions, 101.6 M samples, 868 channels): durable ingestion with search features and chunk summaries, envelope build, read-back of every channel against its SHA-256, interval reads, statistics, overviews and 20 similarity searches. Each run passed only with no out-of-memory kill, no restart, no failed statement, every channel bit-exact and search result lists identical to the reference run's.

Container limits (EX1). On the paper's machine, three runs each without limits, at 2 CPUs and 1 GB, and at 2 CPUs and 512 MB passed [28]. Rates were within 1% of the unlimited runs (1.60 against 1.62 M samples/s ingestion; search 29 ms at p50). PostgreSQL's process memory stayed near 16 MiB; at 512 MB the kernel reclaimed page cache at the limit about 8,000 times per run without an out-of-memory event or a measured slowdown. One ingest connection used about one core, so two cores did not limit it.

A Raspberry Pi 3B+ (EX2). The same image ran with the same 512 MB limits on a Raspberry Pi 3B+ (4 Cortex-A53 cores at 1.4 GHz, 1 GB, a 32 GB microSD card as the only storage, no fan), with the client on the device [29]. All nine runs passed, and search returned the same lists as on the desktop. Without a fan the board ran at its soft temperature limit (1.2 GHz, up to 71 ℃) throughout, with no undervoltage. Medians of three fanless runs: 78,300 samples/s ingestion with features and 258,000 without (about 3 and 10 channels at 25.6 kHz), envelope build 315 s, 4,096-sample reads 6.4 ms and top-ten search 613 ms at p50; the container peaked at about 120 MiB. Against the desktop, ingestion was 20× and search 21× slower. Two deviations were recorded before the affected results were read: the first round ran with a desk fan (reported separately; 20% faster ingestion with features) and the cool-down gate before each run was raised from 55 to 60 ℃. We make no claim about other devices, long-term operation or power loss.

8 Agent access over MCP (case studies)

MorletDB ships a read-only Model Context Protocol server with twelve tools (`list_collections`, `coverage`, `list_recordings`, `trend`, `locate_time`, `describe_interval`, `envelope_spectrum`, `compare_intervals`, `find_onsets`, `search_from_time`, `search_similar`, `read_interval`) [33]. Each call runs in a read-only transaction with a statement timeout, as the database role it is configured with, so ordinary privileges and the row-level security of Section 6 decide what an agent can see; database errors are returned to the agent as tool errors it can correct. Unit tests check tool registration, read-only annotations and error reporting without a database; a stdio smoke test runs the tools against one. The following are case studies, not evaluations.

Seismic investigation. On the paper's development data, an agent asked to find the M2.6 earthquake offshore Tarapacá on 14 March 2025 at station PB01 and to search for similar signals located the time, described the window and searched across four stations and two weeks. Three of the five closest intervals contained other catalogued earthquakes (M2.7–3.0); two had no catalogue entry. The catalogue was joined afterwards for the write-up; the server does not know it. Transcripts of two recorded sessions with Claude on a public four-day subset, run from a Docker-only demo, are in the repository. E5 and E5-H measure the underlying question (Section 10).

Machine demo. A second Docker-only demo loads the IMS bearing run-to-failure tests 2 and 3 [14] (8,152 recordings, 167 M samples, 73,368 search windows) for the same server; recorded sessions follow an agent dating the onset of change in the failing bearings from trends, and comparing fault signatures across the two tests with envelope spectra. The failure labels are not in the

database, and the sessions are illustrations of the workflow, not detection results.

9 Study overview

Confirmatory studies committed their protocol (questions, data, arms, metrics, decision rule) to the repository before their first result; exploratory diagnostics fixed a protocol without a confirmatory decision, or had none (pruning); the grid analysis and the choice of illustrative cases are post hoc. Protocols are internal plans with commit times set by the author's machine, not registrations with a third party. E1 re-measured data already used for BL1-C's confirmation and is not another independent confirmation. The full table with questions, results and limitations is `research/paper/STUDIES.md`.

10 Application evidence and negative results

Catalogued earthquakes (E5, E5-H). For each catalogued EMSC event of magnitude at least 2.5 within 2° of a station, the window containing its predicted P arrival (iasp91) was the query, and a hit counted as relevant if it contained the predicted P arrival of any catalogued event. Both arms used the same exact search with the product's ordering, exclusion and overlap suppression, differing only in the lag range ($L = 30$ against $L = 0$). On development data (465 queries from 171 events) precision at ten was 0.297 against 0.213, a difference of +0.084 with 95% interval $[0.070, 0.098]$ by resampling events; on held-out stations and year, whose waveforms earlier studies had searched but whose labels were new (116 queries from 92 events), 0.139 against 0.060, +0.078 $[0.054, 0.105]$. Bounded lag was higher at every station of both sets, chance was 0.004 and 0.003, and the two top tens shared about half their windows [32]. The frozen rule read "bounded lag better" both times. Catalogue membership is a proxy that misses uncatalogued events and counts invisible small ones, which affects both arms alike; the comparison, not the level, is the result.

Sleep staging by example (S1). On 153 Sleep-EDF nights with a subject-disjoint split, staging each 30-second epoch by the 40 most similar expert-scored epochs from other subjects gave a mean per-subject Cohen's κ of 0.583, against 0.541 for logistic regression on the same features (+0.042, 95% CI $[0.016, 0.068]$), with equal macro-F1 [20]. Retrieval thus exposes source-linked neighbours at no measured accuracy cost against this simple comparator; part of the κ difference comes from the regression's class-balanced weights. The result is far from state-of-the-art staging and makes no clinical claim.

Morlet versus FFT. In the same study, FFT band powers with the same band layout gave κ 0.589 against 0.583 for Morlet energies. We found no advantage for the wavelet representation and made the extractor replaceable.

Wavelet and progressive storage. A compression benchmark found the tested wavelet decomposition codec 19–57% *larger* than scalar quantization with conventional coding at matched error bounds, and progressive raw storage missed its byte-saving gate (0.59 against ≤ 0.5 of a cascade) because a separate envelope index dominated [21]. This motivated storing a retrieval representation *next to* losslessly coded raw data rather than a multiresolution raw format.

Reference coding. Coding a machining recording as a residual against an earlier execution of the same operation would have saved at most 0.0019% on the Bosch CNC data [12], because executions were too dissimilar sample by sample [34].

Out of scope. Three later studies tested retrieval-based forecasting of bearing and turbofan degradation and current-to-vibration estimation; none passed its gate [15]. They concern analytics on top of the store rather than storage and retrieval.

Licence. © 2026 Seung-Zin Nam. Text and figures: CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/); source code: Apache-2.0.

References

[1] Azim Afroozeh, Leonardo X. Kuffo, and Peter Boncz. 2023. ALP: Adaptive Lossless floating-Point Compression. *Proc. ACM Manag. Data* 1, 4, Article 230 (Dec. 2023), 26 pages. doi:10.1145/3626717.

[2] Tony Robinson. 1994. SHORTEN: Simple lossless and near-lossless waveform compression. Technical Report CUED/F-INFENG/TR.156, Cambridge University Engineering Department.

[3] Martijn van Beurden and Andrew Weaver. 2024. Free Lossless Audio Codec (FLAC). RFC 9639, RFC Editor. doi:10.17487/RFC9639.

[4] Robert F. Rice. 1979. Some Practical Universal Noiseless Coding Techniques. JPL Publication 79-22 (NASA-CR-158515), Jet Propulsion Laboratory.

[5] Bob Kemp, Alpo Värri, Agostinho C. Rosa, Kim D. Nielsen, and John Gade. 1992. A simple format for exchange of digitized polygraphic recordings. *Electroencephalography and Clinical Neurophysiology* 82, 5, 391–393. doi:10.1016/0013-4694(92)90009-7.

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

[7] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proc. 2019 Int. Conf. on Management of Data (SIGMOD '19)*, 1981–1984. doi:10.1145/3299869.3320212.

[8] Bob Kemp, Aeilko H. Zwinderman, Bert Tuk, Hilbert A. C. Kamphuisen, and Josefien J. L. Oberyé. 2000. Analysis of a sleep-dependent neuronal feedback loop: the slow-wave microcontinuity of the EEG. *IEEE Trans. Biomed. Eng.* 47, 9, 1185–1194. doi:10.1109/10.867928.

[9] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. 2000. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation* 101, 23, e215–e220. doi:10.1161/01.CIR.101.23.e215.

[10] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I. Lunze, Wojciech Samek, and Tobias Schaeffter. 2020. PTB-XL, a large publicly available electrocardiography dataset. *Scientific Data* 7, Article 154. doi:10.1038/s41597-020-0495-6.

[11] 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.

[12] 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.

[13] Christian Lessmeier, James Kuria Kimotho, Detmar Zimmer, and Walter Sextro. 2016. Condition Monitoring of Bearing Damage in Electromechanical Drive Systems by Using Motor Current Signals of Electric Motors: A Benchmark Data Set for Data-Driven Classification. In *Proc. European Conf. of the PHM Society (PHME 2016)*. doi:10.36001/phme.2016.v3i1.1577.

[14] J. Lee, H. Qiu, G. Yu, J. Lin, and Rexnord Technical Services. 2007. Bearing Data Set. IMS, University of Cincinnati. NASA Prognostics Data Repository, NASA Ames Research Center. https://data.nasa.gov/dataset/ims-bearings.

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

[16] MorletDB contributors. Plain PostgreSQL baselines P1 and P2. `evaluation/plain_postgres/README.md` in [15].

[17] MorletDB contributors. B1: lossless storage against modern baselines. `evaluation/storage_baselines/` in [15].

[18] MorletDB contributors. R1b confirmation (2022). `evaluation/materialized/compact/results/conf-2022/RESULTS.md` in [15].

[19] MorletDB contributors. R2: retrieval scaling, and R2-D diagnostic. `evaluation/retrieval_scaling/` in [15].

[20] MorletDB contributors. S1: sleep staging by retrieval. `evaluation/sleep/RESULTS.md` in [15].

Study	Question	Protocol	Results	Data	Status
R1b	float16 envelopes precise enough?	bfdf6fe 10-26 17:41	db63d58 19:16	unread 2022 slice	confirmation
R2	exact scaling; zero-lag first stage?	12cad82 10-31 14:20	c07396e 14:55	99.9% unread	corrected by R2-D
R2-D	ANN error or candidate metric?	7a7b276 15:57	246c91a 16:24	R2's	exploratory; reuse
BL1	shifted probes, configuration	a5381d7 16:44	e831e51 17:29	R2's	development
BL1-C	frozen configuration on unread data	cc99d09 17:31	635ebd4 18:01	unread stations, year	confirmation
BL1-S	maintenance at commit	ceb63ad 17:33	0d5df8b 18:19	R2's	development
SS1	one stored copy enough?	1942b83 19:20	6a34b63 20:03	R2's, BL1-C's	reuse; bounds missed
SS3	filters and routing	71fb9ae 20:21	136c81a 20:54	R2's, BL1-C's	reuse; corrected (SS3b)
Pruning	exact pruning $\geq 2\times$?	—	9aea409 10-30	2022 slice	exploratory; stopped
Grid analysis	why spacing five?	—	0ef13d1 11-01	dev	post hoc
C2, C3	target ingest; 2× overload	5ad1f12 11-01 09:32	5fadadb 10:56	both	C3 failed
C2p, C4	tuned configuration; 2 hours	2412a77 10:57	167feb6 (#76)	both	C4 aborted
E1	all arms, one build	34904f6 12:12	9c2b6b0 13:26	dev, BL1-C's	re-measurement
D1, D2	time per stage; correlated filters	fa367ed 13:02	e335217 13:32	dev	exploratory
E5	catalogued earthquakes, bounded vs zero lag	2daae97 11-02 10:45	9b8842b 10:47	dev, new labels	reuse; confirmatory
E5-H	E5 on held-out stations and year	8c0c4ba 12:29	fad1659 12:32	BL1-C's, new labels	replication
B3	storage, ingest, reads vs QuestDB	5de3636 18:14	0270be3 19:17	Paderborn	stage 1
B3.1, B3.2	client path, SQL 0.3.3; amendments	in protocol	4f8c8e7, e0c4839	dev set; benchmark set	confirmed once
G1	grid detection without a quantum	e7f6a5b 21:33	ba69f6f, 27f6089	B1 inputs, B3 set	confirmatory
CS1 (S1)	chunk summaries: faster, identical statistics?	5c6d053 11-03 01:48	ad8ce41 02:14	B3 dev set; dev	confirmatory
B3.3	acquisition reads; cost of features at ingest	bf8d3fe 10:51	23b7e9d 14:13	dev set; benchmark set	confirmed once
B3.4	CrateDB 6.4, three layouts	6996513 11-04	9f8c668 19:25	dev set; benchmark set	1 deviation (rerun)
B3.5	Apache IoTDB 2.0, two layouts	fa2c0a0 20:40	50cd12d 21:18	dev set; benchmark set	confirmatory
B3.6	TDengine 3.4, three layouts	6a90427 22:37	de3c790 23:29	dev set; benchmark set	1 deviation (tsz lossy)
B3.7	TimescaleDB 2.31, rowstore and columnstore	9953afd 23:52	0db4c60 11-05 00:19	dev set; benchmark set	confirmatory
B3.8	QuestDB beside the released MorletDB, one session	89a9180 00:39	ce08767 03:30	benchmark set	busy first session set aside
B3.9	B3.4–B3.7 re-paired in one session	2543976 01:07	ce08767 03:30	benchmark set	supersedes B3.4–B3.7 speed
B3.10	next release (I1): one-connection ingestion	a388f87 10:07	f1b46c9 10:27	dev set; benchmark set	read criterion not met
B3.11	B3.10's read criterion, three loads per arm	82b8b6d 10:40	1bfe6f5 10:53	benchmark set	not met; investigated
I1 reads	cause of B3.11's read difference	—	0d08ed4 12:07	benchmark set	exploratory
B3.12	B3.11 with fresh-process and server-side reads	d819edb 12:16	fd82e9d 12:28	dev set; benchmark set	met; I1 adopted
EX1 (X1)	full workflow at 2 CPUs, 512 MB	5a04fe9 14:33	e354488 22:42	B3 benchmark set	confirmatory
EX2 (X2)	full workflow on a Raspberry Pi 3B+	b927fa0 11-04 00:24	d072dff 18:01	B3 benchmark set	2 deviations
B4	codec vs 2026 float codecs	ce3084e 09:25	15edc67 09:31	B1 inputs; Elf series	confirmatory; reuse

[21] MorletDB contributors. Stage-1 compression benchmark. `evaluation/compression/results/ab70819/RESULTS.md` in [15].

[22] MorletDB contributors. BL1: shifted-probe retrieval, confirmation and ingestion. `evaluation/bounded_lag/` in [15].

[23] MorletDB contributors. B2: ClickHouse and TileDB storage baselines. `evaluation/storage_b2/RESULTS.md` in [15].

[24] MorletDB contributors. Pilot stages 1–2: access control, recovery and load. `evaluation/pilot/RESULTS.md` in [15].

[25] MorletDB contributors. CW1: external compute workers. `evaluation/external_worker/CW1_RESULTS.md` in [15].

[26] MorletDB contributors. B4: the chunk codec against the 2026 float codecs. `evaluation/storage_b4/RESULTS.md` in [15].

[27] MorletDB contributors. CS1 (S1 in the repository): per-chunk summaries. `evaluation/chunk_summaries/RESULTS.md` in [15].

[28] MorletDB contributors. EX1 (X1 in the repository): the full workflow inside edge-sized container limits. `evaluation/edge_x1/RESULTS.md` in [15].

[29] MorletDB contributors. EX2 (X2 in the repository): MorletDB on a Raspberry Pi 3B+. `evaluation/edge_x2/RESULTS.md` in [15].

[30] MorletDB contributors. G1: automatic calibration-grid detection. `evaluation/grid_detection/RESULTS.md` in [15].

[31] 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 next release (B3.10–B3.12). `evaluation/tsdb_b3/RESULTS.md` in [15].

[32] MorletDB contributors. E5 and E5-H: catalogued earthquakes, bounded-lag against zero-lag search. `evaluation/task_relevance/` in [15].

[33] MorletDB contributors. Read-only MCP server for AI agents, agent and machine demos. `tools/mcp/`, `docs/mcp.md`, `examples/` in [15].

[34] MorletDB contributors. X1: reference coding across repeated operations. `evaluation/compression/reference/RESULTS.md` in [15].