Storage-Level Pitfalls in Time-Series Database Engines: When Compression Hurts and Why

Fuming Fu
Independent Researcher
Austin, TX, USA
mingming86x@gmail.com
ORCID: 0009-0001-6728-2677

Abstract—Time-series database (TSDB) benchmarks evaluate query latency and ingest throughput but overlook storage-level costs that can dominate total cost of ownership. We present the first cross-engine characterization of write amplification (WA), space amplification (SA), and compression ratio (CR) across four TSDB storage architectures—columnar merge-tree, heap-with-chunk-compression, append-only column files, and Parquet object storage—on synthetic, autocorrelation-controlled, and real-world datasets at scales up to 500 M rows. Our results reveal three storage-level factors invisible to existing benchmarks. First, enabling compression on undersized chunks increases storage by up to 54%, with a non-monotonic crossover zone below ∼2,800 rows per chunk; the pathology persists into TimescaleDB 2.27's Hypercore-era columnstore and recurs in VictoriaMetrics via label-encoding overhead, while ClickHouse parts stay CR>1—the overhead-floor mechanism is paradigm-level, not version- or engine-specific. Second, data characteristics (value precision, schema design) affect compression 7.6–13× more than codec selection (1.3–1.6×), and a controlled experiment shows that numeric autocorrelation alone has zero effect on full-precision floats; the ALP codec narrows but does not invert this ratio. Third, ingest batch size controls write amplification by up to 6.6× through its effect on background merge volume, with the effect persisting across codec configurations. A query-latency snapshot at 10M rows shows that the best-CR codec configuration is also the best tail-latency configuration, dispelling the assumption of a storage-vs-query tradeoff. We derive four evidence-based configuration guidelines from these findings.

I. Introduction

Enabling TimescaleDB compression on a table with 100,000 rows and a one-hour chunk interval increased disk usage from 73 MB to 112 MB—a 54% storage penalty. This counterintuitive result is specific to TimescaleDB's TOAST-based compressed-row layout (we trace the cause in §IV-A), but it is invisible to standard TSDB benchmarks such as TSBS [1], which measure only query latency and ingest throughput. More broadly, we find that storage-level pitfalls – compression that hurts, codec choice that is dominated by data shape, and batch size that silently controls write amplification – recur across engines and dominate cost in ways query benchmarks cannot see.

Modern TSDBs span columnar merge-tree (ClickHouse [2]), heap+chunk compression (TimescaleDB [3]), append-only column files (QuestDB [4]) or label-indexed metric series (VictoriaMetrics [5]), and Parquet object storage (InfluxDB 3 Core [6]). Each makes fundamentally different write-amplification (WA), space-amplification (SA), and compression-ratio (CR) tradeoffs. On cloud block storage ($0.08/GB-month), an SA difference of 2× vs. 5× on 10 TB costs $1,600 vs. $4,000 per month, yet no published study measures these storage-level metrics across engines under controlled conditions.

We address this gap with the first cross-engine, storage-internal characterization of TSDB engines. Our contributions are:

1) A measurement methodology using decomposed storage metrics (WA into initial-writes and merge-writes; SA into data, index, and metadata; CR per column type) validated across four storage paradigms with five engine implementations (ClickHouse, TimescaleDB, QuestDB, VictoriaMetrics, InfluxDB 3) at scales from 100K to 500M rows.
2) Three empirical findings that challenge common configuration assumptions: compression can increase storage (and the pathology persists into TimescaleDB 2.27's Hypercore-era columnstore conversion), codec selection is secondary to data characteristics (precision, schema; the ALP codec narrows but does not invert this ratio), and batch size is the dominant controllable factor for write amplification.
3) Four evidence-based configuration guidelines derived from two datasets (synthetic network telemetry and real-world NOAA climate observations [7]), supported by a query-latency snapshot that shows storage-favourable configurations also win on tail latency.

II. Background and Related Work

A. Storage Paradigms

Table I summarizes the four architectures studied as representative implementations of four commonly cited TSDB storage paradigms. We do not claim exhaustive coverage: other engines (e.g., TDengine, expected to be a variant of the columnar merge-tree) plausibly map to

TABLE I
Storage paradigms studied.

Paradigm	Engine	Writes	Compr.
Columnar merge-tree	CH	Part→merge	Per-col.
Heap+chunk compr.	TS	WAL→heap→col.	Type-spec.
Append-only columns	QDB,VM	WAL→mmap	None
Parquet object store	IOx	WAL→Parquet	LZ4

one of these paradigms, but verifying those mappings experimentally is left to future work.

The columnar merge-tree (ClickHouse MergeTree) writes each INSERT as an immutable sorted part that is later merged in the background, applying per-column codecs (LZ4, DoubleDelta, Gorilla, ZSTD). The heap-with-chunk-compression paradigm (TimescaleDB) partitions data into time-based chunks stored as PostgreSQL heap tables; an explicit compress_chunk() call converts heap rows into a columnar TOAST representation (or, in v2.18+, a Hypercore columnstore conversion that we measure separately) using type-specific algorithms (Gorilla for floats, delta-of-delta for timestamps). The append-only column files paradigm (QuestDB) memory-maps one file per column per partition with no compression, optimizing for write throughput. The Parquet object-storage paradigm (InfluxDB 3 Core [6], the open-source successor to InfluxDB IOx) buffers writes in a WAL, flushes Arrow record batches to immutable Parquet files on a backing object store (local disk in our setup), and relies on a background compactor for steady-state SA.

B. Related Work

TSDB benchmarks—TSBS [1] (insert rate, query latency p50/p99) and IoTBench [8] (throughput under mixed workloads)—report performance metrics but measure no storage internals (WA, SA, CR). Compression research evaluates codecs in isolation: Gorilla [9] introduced XOR-based float compression, ALP [10] achieves state-of-the-art lossless float compression via decimal detection, and BtrBlocks [11] auto-selects column codecs for data lakes. TerseTS [12] provides a standardized evaluation framework for time-series compression algorithms. At the storage-systems level, MSST 2024 characterized HDD seek behavior [13], LSM compaction on ZNS SSDs [14], and MLaaS I/O patterns [15]. Write amplification has been studied in LSM-tree key-value stores [16], but no work targets TSDB storage engines specifically.

III. Methodology

We propose a measurement framework for TSDB storage internals built on three principles: (1) decomposed metrics that reveal where bytes go (not just how many), (2) a two-state protocol that captures both operational and steady-state behavior, and (3) controlled single-variable experiments that isolate each factor.

A. Configurations

We evaluate seven primary configurations spanning four paradigms: CH-Default (ClickHouse, LZ4), CH-Optimized (ClickHouse, DoubleDelta/Gorilla/Delta+ZSTD per-column), TS-1day (TimescaleDB, 1-day chunks), TS-1hour (TimescaleDB, 1-hour chunks), QDB-Raw (QuestDB, no compression), IOx-Default (InfluxDB 3 Core, LZ4 Parquet), and VM-Default (VictoriaMetrics [5], default label encoding). Extended experiments add NOAA-specific configurations (4 variants), batch-size variants from bs=1 to bs=50,000, a TimescaleDB chunk-interval sweep at 7 intervals (10 min–7 d), a Hypercore-era columnstore variant at 4 crossover volumes (TS 2.27.0), an ALP codec measurement on NOAA Float64 columns, CH-Default and CH-Optimized scale runs to 500M rows, and crossover-volume runs at 13 volumes, totaling more than 50 distinct configurations.

B. Metrics

We measure three decomposed metrics:

- Write Amplification (WA) = total physical bytes written / logical data ingested. For ClickHouse, decomposed via system.part_log into initial-write WA and merge WA. For TimescaleDB, measured via pg_stat_wal deltas. For InfluxDB 3 and VictoriaMetrics we report on-disk bytes / logical bytes as conservative WA proxies; the upstream compactor's rewrite volume is not exposed by either API, so these numbers bound but do not match the per-engine ClickHouse and TimescaleDB WA and are reported with explicit caveats (§V).
- Space Amplification (SA) = bytes on disk / logical data size. Logical row size = 75 B (sum of column widths in the synthetic schema, engine-independent). Decomposed into data, index, and TOAST/metadata components.
- Compression Ratio (CR) = uncompressed / compressed bytes, from system.parts. Per-column breakdown from system.parts_columns (ClickHouse only; TimescaleDB exposes only table-level CR—itself a finding).

C. Datasets

Synthetic: 11-column network telemetry (2 Float64 drawn i.i.d. from $\mathcal{N}(\mu, \sigma^2)$, 5 integers, 2 strings, 1 timestamp). This represents a worst case for XOR-based codecs because consecutive float values are uncorrelated.

Autocorrelation controls: Two additional synthetic variants with identical schema but random-walk floats (lag-1 r=0.999, consecutive diff ∼1.2) and moderate-walk floats (r=0.975, diff ∼5.4). These isolate the autocorrelation effect from schema differences.

Real-world: NOAA Global Historical Climatology Network [7], 1.07 billion rows spanning 120 years. Float

columns (temperature, pressure) are strongly autocorrelated and quantized (0.1 °C precision)—the expected best case.

D. Protocol

All engines run in Docker containers (2 CPU, 4 GB RAM) on an Intel i5-7300HQ (4 cores, 16 GB RAM, SATA SSD) under Ubuntu Linux. Engine versions: ClickHouse 25.3, TimescaleDB 2.17 (legacy compress_chunk; covers all main results), QuestDB 8.2, VictoriaMetrics (latest 2026-05), InfluxDB 3 Core (latest 2026-05). Hypercore-era validation (§V) uses TimescaleDB 2.27.0 separately.

Each experiment follows a nine-step protocol: (1) deploy fresh container, (2) create schema, (3) reset baselines (part-log marker, WAL position), (4) ingest data in batches, (5) EWMA-based convergence detection (α=0.3, ϵ=0.001, K=5 stable samples) with engine-specific hard gates (merges = 0 for ClickHouse, no autovacuum for TimescaleDB), (6) measure fresh state, (7) run maintenance (OPTIMIZE TABLE FINAL or compress_chunk), (8) re-converge, (9) measure optimized state. We run 3 seeds per configuration (Table II and the early crossover-volume sweep include a 4th seed at four volumes); we report means in §IV and medians in §V where seed-level tail effects matter.

Validation. We verify the framework via three independent checks: (1) SA is batch-invariant (0.6206 ± 10^{-8} across 7 batch sizes), confirming that batch size is a WA-only parameter as the framework predicts—if SA varied, our metric decomposition would be suspect; (2) cross-seed CoV <1% for 83% of CR measurement groups; and (3) the TS-1day control produces CR >1 for all 21 records (minimum 2.31), confirming the rows-per-chunk mechanism independently of the TS-1hour experiments.

IV. Results

A. Compression Can Increase Storage

Figure 1 plots CR as a function of rows per chunk for TimescaleDB with 1-hour and 1-day chunk intervals across 13 data volumes (100K–10M rows). With 1-hour chunks (720 chunks from a 30-day span), CR oscillates below 1.0 up to ∼2,800 rows per chunk: compression increases disk usage. At 139 rows/chunk (100K rows), CR = 0.82 and SA rises 54% after compression is enabled. The crossover zone is non-monotonic—CR briefly exceeds 1.0 near 1,000 rows/chunk before dipping to 0.76 at 1,736 rows/chunk (Table II). Compression reliably helps only above ∼2,800 rows/chunk.

The root cause is architectural, not version-specific. TimescaleDB compression converts N heap rows into a single compressed row of per-column TOAST arrays in fixed batches of 1,000 rows. Each compressed row carries fixed overhead: PostgreSQL heap header (23 B), one TOAST pointer per column (∼18 B × N_{col}), min/max metadata for the orderby column, and at least one 8 KB TOAST page per column. For our 11-column schema this yields ≥22 KB per compressed batch, which exceeds the data content when a chunk holds only 139 rows (10 KB logical). TOAST page allocation is discrete: our measurements show that volumes 1.25M, 1.5M, and 1.75M all produce identical 631 KB compressed chunks despite 40% variation in uncompressed data, confirming the step-function behavior. The crossover threshold depends on column count and row width (more columns ⇒ higher overhead ⇒ higher threshold); the existence of a crossover is fundamental to any system that stores compressed data in per-column TOAST arrays with fixed batch sizes.

TABLE II
Crossover detail for TS-1hour (optimized state, 3–4 seeds). CR oscillates below 1.0 up to ∼2,800 rows/chunk.

Volume	Rows/Chunk	CR	Effect
100K	139	0.821	× (+54%)
200K	278	0.805	× (+46%)
500K	694	0.835	× (+32%)
750K	1,042	1.022	≈
1M	1,389	1.026	≈
1.25M	1,736	0.759	× (+32%)
1.5M	2,083	0.910	× (+10%)
2M	2,778	1.121	✓ (−11%)
5M	6,944	1.268	✓ (−21%)
10M	13,889	2.262	✓ (−56%)

As a control, we repeat the experiment with 1-day chunks (30 chunks per 30-day span). Even at 200K rows, TS-1day yields 6,667 rows per chunk and achieves CR = 2.84—always safely above 1.0. At the same 200K volume, TS-1hour has 278 rows/chunk and CR = 0.81: a 3.5× difference from chunk interval alone. Chunk granularity also impacts ingest: TS-1hour sustains 9,600 rows/s at 10M but degrades to ∼200 rows/s beyond 18M rows as the 720-chunk working set exceeds PostgreSQL's shared buffer pool.

Sweeping chunk interval at fixed 10M rows (squares in Fig. 1) confirms the trend in the other axis: across seven intervals (10 min–7 d) CR rises monotonically from 1.93 to 4.05 post-crossover. WA, by contrast, is non-monotonic: it reaches a minimum of 9.6 at 30 min chunks (vs. 15.7 at 10 min and 14.5 at 7 d), so the CR maximum and the WA minimum live in different chunk-interval regimes—a tradeoff G2 makes explicit.

Practitioner impact. Any configuration below ∼2,800 rows per chunk—onboarding, dev/staging, or low-rate-metric long-tail workloads—silently inverts compression's purpose: storage rises up to 54% with no engine-side warning. This is a routine misconfiguration trap, not an academic edge case.

B. Data Characteristics Dominate Codec Selection

Figure 2 compares CR between the synthetic dataset and the NOAA climate dataset across ClickHouse configurations at 10M and 50M rows. At 10M, switching from synthetic to NOAA data improves CR by 7.6× (CH-Default)

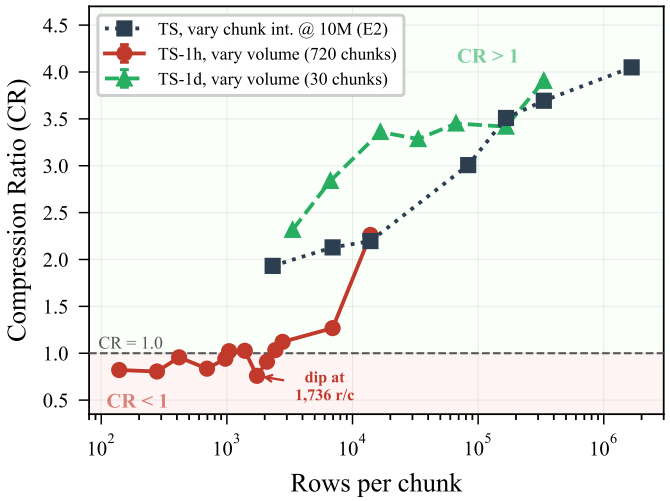

Fig. 1. CR vs. rows/chunk for TimescaleDB. Circles (TS-1h) and triangles (TS-1d) sweep total rows. Squares hold rows at 10M and sweep chunk_interval across seven values, showing monotonic post-crossover CR gain up to 1.67M rows/chunk.

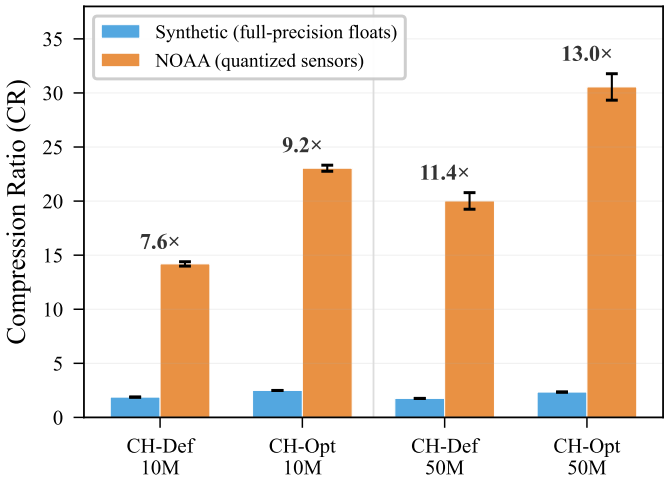

Fig. 2. Compression ratio: synthetic vs. NOAA data at 10M and 50M rows. Data characteristics drive 7.6–13× CR differences; codec selection (the small gap between CH-Def and CH-Opt blue bars) contributes only 1.3–1.6×. A controlled random-walk experiment (§IV-B) confirms the gap is driven by value precision, not autocorrelation.

to 9.2× (CH-Optimized). Switching from default codecs (LZ4) to optimized codecs (DoubleDelta/Gorilla/ZSTD) improves CR by only 1.33× on synthetic data and 1.62× on NOAA data. At 50M the gap widens: 11.4–13.0× data effect, up from 7.6–9.2× at 10M.

To isolate the autocorrelation effect from schema differences, we generated two additional synthetic variants with the same 11-column schema: a random walk (consecutive float diff ∼1.2, vs. 16.8 for i.i.d.) and a moderate walk (diff ∼5.4). The result: table-level CR is unchanged (LZ4 = 1.88, Gorilla = 2.49 at 10M, all three modes). Per-column analysis confirms that Float64 columns remain at CR≈1.0 regardless of autocorrelation or codec—Gorilla's XOR encoding exploits bit-level identity (shared leading bits in IEEE 754), not numeric proximity, and full-precision floats defeat it.

The 7.6× gap therefore reflects value precision and schema design: NOAA's quantized sensor readings (0.1 °C precision ⇒ many zero mantissa bits), bounded-range integers, and low-cardinality station identifiers—not autocorrelation per se. The practical implication is that practitioners who assume "my data drifts slowly, so Gorilla will help" are likely wrong unless their values are also discretized. The overall data profile—precision, value ranges, cardinality, sort order—drives a 7.6–13× CR difference versus 1.3–1.6× from codec selection, and this gap widens at scale.

Where to optimize first. Vendor docs foreground codec choice (Gorilla, ZSTD, DoubleDelta) as "the" compression decision, but our results put the larger lever upstream of the database: quantize values to needed precision and contain string cardinality before retuning codecs. Schema and precision control yield an order-of-magnitude win that no codec retune can match.

C. Batch Size Controls Write Amplification

Figure 3 shows ClickHouse WA as a function of ingest batch size across seven sizes (500–50,000) at 1M rows. For CH-Default, WA decreases monotonically from 8.03 (batch size 500) to 1.65 (batch size 50,000)—a 4.9× reduction. At 10M the pattern holds: 8.52 to 2.05 (4.2×).

The effect persists across codec configurations: CH-Optimized (DoubleDelta/Gorilla/ZSTD) shows the same monotonic decrease, from WA = 7.42 (bs = 500) to WA = 1.12 (bs = 50,000) at 1M—a 6.6× range across the same six batch sizes.

While write amplification from small batches is a known property of LSM-tree architectures, its magnitude in TSDB engines has not been quantified. We decompose WA into initial-write (data written once, ≈0.65 regardless of batch size) and merge writes (data rewritten during compaction). At batch size 500, merges account for 91.2% of total WA; at 50,000, this drops to 62.2%. SA and CR are completely unaffected by batch size (SA = 0.621 ± 0.0004 across all seven sizes)—it is a WA-only parameter.

Operational cost angle. Batch size is a client-side knob invisible to storage dashboards, yet it sets the long-run device write workload. The 2.3–2.5× WA reduction from G3 buys proportional SSD-life and compaction-CPU savings—a free win that no query-latency benchmark would expose.

D. Query Latency Snapshot

The findings above target storage cost, but practitioners need to know whether favourable storage configurations carry hidden query-side penalties. Table III reports post-warmup p50/p99 latency across five representative queries (time-range aggregate, filtered scan, high-cardinality group-by, top-k, latest-window) at 10M rows,

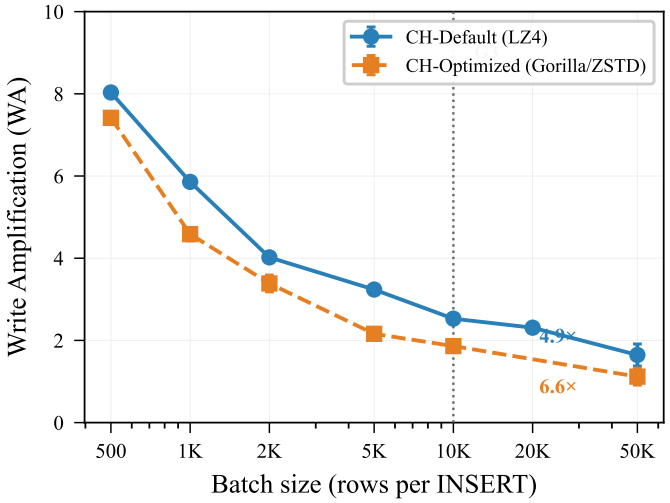

Fig. 3. Write amplification vs. batch size for CH-Default (7 sizes) and CH-Optimized (6 sizes) at 1M rows. Both codecs show monotonic WA decrease (4.9× and 6.6× full range).

TABLE III
Query p50/p99 (ms) at 10M rows, two seeds, 30 iter/query. Best per column in bold. TS-1hour Q6 p99 >9 s arises from per-chunk planning overhead across 720 chunks.

Config	Q1 agg	Q3 scan	Q4 grp	Q5 top-k	Q6 latest
CH-Default	9.8/41	13.4/28	11.4/31	49.1/119	26.4/31
CH-Optimized	11.9/19	13.6/17	12.9/15	56.3/84	51.6/62
TS-1hour	54.4/601	25.8/99	18.9/20	360/673	797/9652
TS-1day	29.2/656	3.3/24	27.2/32	502/583	21.6/134
QuestDB	13.7/45	16.1/84	16.6/36	42.8/540	329/658

two seeds, 30 iterations per query. Three observations resolve the concern. First, CH-Optimized—the per-column codec configuration that delivers the best CR (Fig. 2)—also delivers the best tail latency: p99≤84 ms on every query. Selecting CR via codec tuning does not trade away query latency. Second, TimescaleDB chunk size affects query latency as strongly as it affects storage: TS-1hour's Q6 (latest-window) p99 of 9.6 s collapses to 134 ms at TS-1day, a 72× improvement matching G1/G2's direction. Third, QuestDB's append-only paradigm is competitive on small range scans (Q3 p50 16 ms) but exhibits heavy tail blowup on top-k and latest-window (p99 540–658 ms), consistent with its lack of secondary indexes.

Resolving the storage-vs-query concern. The common "tune for storage, pay at query time" assumption does not hold in the regime we tested: on the two dimensions measured (chunk size, codec), the knobs that improve storage also improve tail latency. Storage and query-tail-latency collapse into a single optimization, not the two-axis tradeoff often cited.

V. Discussion

A. Configuration Guidelines

Our results yield four evidence-based guidelines:

G1: Ensure chunks contain ≫1,000 rows before enabling TimescaleDB compression. The per-batch TOAST overhead scales with column count; for our 11-column schema the safe threshold is ∼2,800 rows/chunk. Wider schemas require proportionally more rows. This threshold is architectural (TOAST storage + fixed batch size), not version-specific (§IV-A).

G2: Tune chunk interval before selecting codecs. At 200K rows, switching from 1-hour to 1-day chunks improves CR from 0.81 to 2.84—a 3.5× difference from interval alone, versus 1.3–1.6× from codec selection. Excessive granularity also degrades ingest throughput by 48× at scale (§IV-A) and query-side latency: at 10M rows the latest-window p99 collapses from 9.6 s on TS-1hour to 134 ms on TS-1day (§IV-D).

G3: Use batch size ≥10,000 for ClickHouse ingest. Compared to the default of 1,000, this reduces WA by 2.3× (CH-Default) to 2.5× (CH-Optimized), with no impact on SA or CR (§IV-C).

G4: Profile value precision before codec selection. Gorilla's XOR compression requires bit-level identity, not just numeric proximity; our controlled experiment shows zero CR gain from autocorrelation on full-precision Float64. Quantized sensor values, bounded integers, and low-cardinality strings drive the 7.6–13× data advantage over codec tuning (1.3–1.6×) (§IV-B).

B. Limitations

Scale. ClickHouse runs reach 500M rows and the CR plateau holds (§IV-B); TimescaleDB runs stop at 10M because TS-1hour ingest collapses beyond that point on our 4 GB container. Cluster-scale and multi-TB distributed validation remain open.

Environment. All engines run in Docker (2 CPU, 4 GB RAM) on a single SATA-SSD host; we report relative comparisons under identical constraints, but absolute thresholds (e.g., the buffer-pool collapse point) will shift on production-class hardware.

Datasets. Two datasets bracket the autocorrelation spectrum but do not cover financial tick data, high-cardinality log telemetry, or string-dominated event streams, where the codec-vs-data ratio of §IV-B could look different.

Workload. The §IV-D snapshot is a single 30-iteration pass on five canonical queries; mixed read/write workloads, concurrent tenants, and large-cardinality joins are not exercised.

Engines and versions. Findings rest on architectural properties (TOAST/columnstore overhead, LSM merge, hypertable routing, Parquet flush, metric-with-labels indexing), not version-specific behaviour, and we cross-validated each where we could. TimescaleDB 2.27's Hypercore columnstore re-runs of the crossover regime yield

CR=0.82, 0.84, 0.63, and 1.08 at 100K/500K/1M/2M rows (the 100K and 500K values match legacy TOAST within 2%; the 1M value falls below legacy's 1.03, plausibly reflecting columnstore segment-layout overhead near the threshold), so the crossover pathology survives the post-2.18 storage rewrite. ALP [10] on NOAA Float64 reaches CR≈4.2–4.6 versus Gorilla's ≈1.0—a real codec win, but still below the 7–13× data effect, so G4 holds. VictoriaMetrics shows CR≈0.056–0.075 across 100K–10M (a per-sample label-encoding floor distinct from TimescaleDB's TOAST floor), while ClickHouse at the same thin-batch regime stays at CR=1.65±0.001, confirming the crossover is a family of overhead-floor mechanisms rather than a single engine quirk. InfluxDB 3 Core (Parquet/LZ4) measures CR=2.08 and SA=0.64 at 10M, comparable to CH-Default. Exact thresholds still vary across implementations, and the InfluxDB 3 and VictoriaMetrics APIs do not expose compactor rewrite volume, so our cross-engine WA is bounded only from on-disk inventory.

C. Open Questions and Better Tooling

Where else does the overhead floor appear? We confirmed two distinct mechanisms in two engines (TimescaleDB TOAST/Hypercore; VictoriaMetrics per-sample label encoding). Whether the floor recurs in TDengine, Druid, M3DB, or Cassandra-backed stacks is open—each has a different per-batch metadata footprint, so the crossover volume likely shifts but its existence should not. Our methodology (decomposed WA/SA/CR, EWMA convergence) ports directly.

When does codec choice escape data dominance? ALP narrows but does not invert the 7–13× data effect on our datasets. Whether a future codec or a workload with sufficiently uniform values can flip that ratio is an open empirical question best answered by a corpus instrumented with the value-precision and cardinality features of §IV-B.

Why don't engines warn? The most useful tooling improvement we can identify is an engine-side check at the action that enables compression (TimescaleDB's compress_chunk / SET (timescaledb.compress), or column-codec declaration at table-create time): estimate rows-per-chunk under current ingest rate, compare to a schema-derived overhead floor, and refuse-or-warn below the crossover. No engine we tested surfaced any such signal. A thin external profiler is feasible today; in-engine folding is the longer-term ask.

Cross-engine WA observability. The instrumentation gap above prevents operators from pricing an engine choice in device-lifetime or compaction-CPU terms. An in-engine "bytes-rewritten-by-compactor" counter at the granularity of today's query-latency metrics would close it and is, in our view, the lowest-cost high-impact addition the TSDB ecosystem could make.

VI. Conclusion

TSDB benchmarks measuring only query latency overlook storage costs that dominate total cost of ownership. Across four paradigms, five engines, and scales to 500M rows we surface three factors invisible to existing benchmarks: (1) compression can increase storage by up to 54% on undersized chunks—a crossover reproduced on TimescaleDB 2.27's Hypercore columnstore and recurring in VictoriaMetrics via label-encoding overhead, while absent in ClickHouse parts; (2) data characteristics affect CR 7–13× more than codec selection, autocorrelation alone yields zero gain on full-precision floats, and ALP narrows but does not invert the ratio; (3) batch size controls WA by up to 6.6× across codec configurations. A query-latency snapshot further shows storage-favourable configurations also win on tail latency, eliminating an assumed storage-vs-query tradeoff. Future work: cluster-scale validation and automated cost models for TSDB configuration selection.

AI Disclosure. LLM tools were used for editing assistance during manuscript preparation. All experimental design, code, measurements, analysis, and figures are the authors' own work.

References

[1] Timescale, Inc., "TSBS: Time series benchmark suite," 2021, https://github.com/timescale/tsbs.

[2] ClickHouse, Inc., "ClickHouse documentation," 2025, https://clickhouse.com/docs.

[3] Timescale, Inc., "TimescaleDB architecture," 2025, https://docs.timescale.com.

[4] QuestDB Ltd., "QuestDB documentation," 2025, https://questdb.io/docs.

[5] VictoriaMetrics Inc., "VictoriaMetrics: fast, cost-effective, scalable time-series database," https://github.com/VictoriaMetrics/VictoriaMetrics, 2026, accessed 2026-05.

[6] InfluxData, Inc., "InfluxDB 3 Core: open-source parquet/arrow time-series engine," https://github.com/influxdata/influxdb, 2026, accessed 2026-05.

[7] M. J. Menne, I. Durre et al., "NOAA global historical climatology network – daily (GHCN-Daily)," 2012, https://doi.org/10.7289/V5D21VHZ.

[8] J. Jiang, Q. Zhang, J. Lu, B. Li, and P. Wang, "An empirical benchmark for time series database in IoT applications," in Proc. IEEE BigData, 2021.

[9] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan, "Gorilla: A fast, scalable, in-memory time series database," in Proc. VLDB Endowment, vol. 8, no. 12, 2015, pp. 1816–1827.

[10] A. Afroozeh and P. Boncz, "ALP: Adaptive lossless floating-point compression," in Proc. ACM SIGMOD, 2024, pp. 441–456.

[11] M. Kuschewski, M. Dreseler, M. Bandle, and A. Kemper, "BtrBlocks: Efficient columnar compression for data lakes," in Proc. ACM SIGMOD, 2024, pp. 1–17.

[12] C. E. Muñiz-Cuza, S. K. Jensen, T. L. Klein, S. Bakhtiiarova, M. Boehm, and T. B. Pedersen, "TerseTS: A framework for time series compression," in Proc. EDBT, 2026, pp. 760–763.

[13] S. Olmez, J. Bent, R. Arpaci-Dusseau, and Y. Dai, "Revisiting HDD rules of thumb: 1/3 is not (quite) the average seek distance," in Proc. MSST, 2024.

[14] G. Liu, C. Yang, Q. Yu, C. Guo, W. Xia, and Z. Cao, "Prophet: Optimizing LSM-based key-value store on ZNS SSDs with file lifetime prediction and compaction compensation," in Proc. MSST, 2024.

[15] Q. Zou, Y. Deng, Y. Zhu, Y. Zhou, J. Cai, and S. He, "Dissecting I/O burstiness in machine learning cloud platform: A case study on Alibaba's MLaaS," in Proc. MSST, 2024.

[16] C. Luo and M. J. Carey, "LSM-based storage techniques: A survey," The VLDB Journal, vol. 29, no. 1, pp. 393–418, 2020.