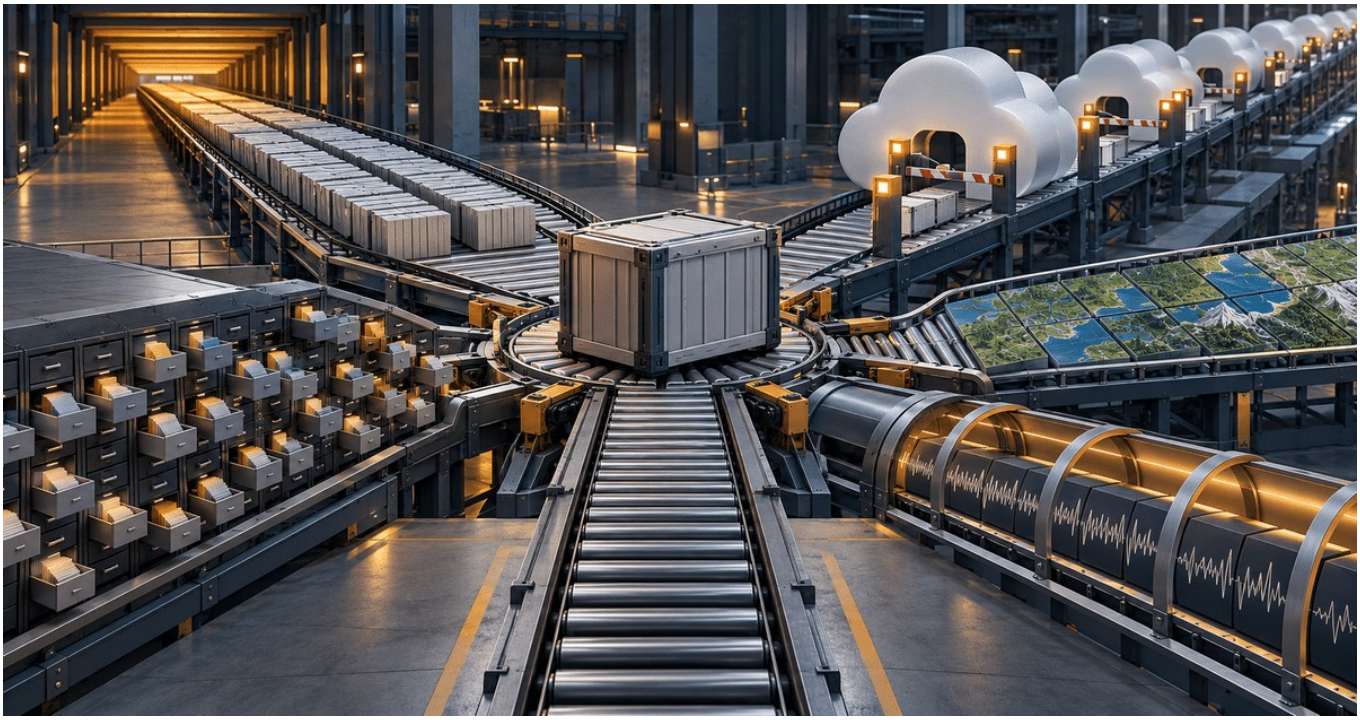
Beyond Parquet and Avro: Choosing a Data File Format by Access Pattern

Julien Laurenceau | Published 2026-10-21 | Updated 2026-10-21

https://pepitedata.com/data-file-formats-beyond-parquet/

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



Parquet has been the default analytical file format for a decade, and Avro the default row format for streaming records. Since 2023 a new generation has appeared around them: Lance, Vortex, Nimble, FastLanes, BtrBlocks and F3 for general data, native geometry types and cloud-optimized formats for geospatial data, and new float encodings for time series. Each one optimizes one access pattern on one kind of storage. This guide maps which pattern, what the published numbers actually measure, and how to choose without betting a platform on a vendor chart.

The one-line version: **choose the format by access pattern and storage tier, measure on your own data with tuned baselines, and keep a copy every reader can open until your table format and engines support the alternative**.

Why are new data file formats appearing now?

New file formats appear because the bottleneck moved. On NVMe drives and fast networks, decoding heavily compressed pages now costs more CPU than the I/O it saves. AI workloads add access patterns Parquet was never tuned for: point lookups for retrieval, projections over thousands of feature columns, and large values such as embeddings and images. Object storage adds a third constraint, where every request costs latency and money.

The clearest independent statement of the shift comes from the 2023 empirical evaluation of Parquet and ORC by Xinyu Zeng, Andrew Pavlo, Wes McKinney, Huanchen Zhang and co-authors, published in PVLDB:

The bottleneck of query processing is shifting from storage to (CPU) computation on modern hardware.

Zeng et al., "An Empirical Evaluation of Columnar Storage Formats", PVLDB 17(2), 2023

The same paper adds the qualifier that matters for capacity planning: on cloud storage, where I/O still dominates, aggressive block compression remains worth its CPU cost. The right format therefore depends on where the bytes live. Its other lessons read like the requirements list of the new formats: metadata that stays cheap on wide feature tables, separate regions for large binary objects, float-specific compression, and encodings a GPU can decode in parallel.

Avro is not part of this race. It is a row-oriented container that writes records in blocks, which still makes it the right shape for Kafka payloads and record-at-a-time writes. Apache Iceberg also uses it for manifest files. The competition is about the columnar formats that analytics and AI pipelines scan.

Where Parquet hurts: four known limits

Parquet stores data in row groups, each split into one column chunk per column and then into pages, with all metadata in a footer that readers parse first. That layout is excellent for large scans. It is weak in four places: point lookups decode a whole page, footer parsing grows with columns times row groups, heavyweight codecs cost CPU at decode, and its variable-length encodings resist SIMD and GPU decoding.

1. **Point lookups.** Fetching one value means reading and decoding the page around it. The Lance authors measured parquet-rs with default settings at 5,500 rows per second on random access to small scalar values.
2. **Wide footers.** The footer is a Thrift structure that cannot be read partially. Zeng et al. measured metadata parsing growing almost linearly with the number of columns, even when a query projects only 10 of them. The Parquet community now has a draft proposal for a FlatBuffers footer, opened in October 2025, motivated by files with thousands of leaf columns from AI datasets.
3. **Codec cost.** Snappy or Zstd on top of the encodings saves bytes and spends CPU. On local NVMe that trade often loses.
4. **Sizing tension.** The Parquet documentation recommends 512 MB to 1 GB row groups and 8 KB pages, while common writers default to much smaller row groups and about 1 MB pages. Large values such as images or embeddings make any single row group size wrong for some column.

Parquet keeps absorbing the new ideas

Parquet is still moving. Format 2.11.0 (March 2025) added VARIANT, GEOMETRY and GEOGRAPHY logical types, and 2.14.0 (September 2026) added the ALP float encoding from CWI's SIGMOD 2024 paper. The catch is reader support: in October 2026 the official Parquet implementation status page lists ALP read support in one of ten implementations, arrow-rs. The spec now moves faster than the engines that read it.

parquet-format release	Date	What it added
2.5.0	March 2018	Column index and offset index (page-level pruning)
2.7.0	September 2019	Bloom filters, modular encryption
2.8.0	December 2019	BYTE_STREAM_SPLIT encoding for floats
2.10.0	November 2023	Float16 type, size statistics

parquet-format release	Date	What it added
2.11.0	March 2025	VARIANT, GEOMETRY and GEOGRAPHY types; BYTE_STREAM_SPLIT for integers
2.12.0	August 2025	Variant shredding finalized
2.14.0	September 2026	ALP float encoding (Preview), FILE logical type

Much of Parquet's reputation comes from default writer settings. In the same paper that introduces Lance, its authors tuned parquet-rs (small pages, page index, compression and dictionary encoding off) and random access went from 5,500 to 350,000 rows per second, more than 60 times faster, at a minor cost in scan speed and memory. The strongest data point for tuned Parquet comes from a competing format's own team.

Two more results point the same way. Arrow Rust 57.0.0 (October 2025) parses Parquet footers 3 to 9 times faster with a custom Thrift decoder and no change to the format. DuckDB 1.2 can write the newer Parquet encodings, which made TPC-H lineitem files 11 to 30 percent smaller in DuckDB's test, but does not write them by default because mainstream engines cannot read them. Some of those encodings are almost ten years old.



Parquet absorbs the ideas of its challengers: geometry and variant types in 2025, ALP float compression in 2026. Reader support follows more slowly.

Before I recommend replacing Parquet, I check how the files were written. A file produced with defaults says little about what the format can do.

The new general-purpose formats, compared

Six designs lead the new generation. BtrBlocks and FastLanes are research formats that replace heavyweight codecs with cascades of lightweight, SIMD-friendly encodings. Lance targets random access for AI datasets, Nimble targets very wide feature tables at Meta, Vortex aims to be the

general-purpose successor to Parquet, and F3 embeds WebAssembly decoders in each file so that old readers can decode new encodings.

Format	Origin	Design center	Status in October 2026	Headline figure (who measured)
BtrBlocks	TUM, SIGMOD 2023	Lightweight encodings chosen by sampling, applied in cascades	Research; approach reused in Vortex	Scans 2.2x faster and 1.8x cheaper than Parquet on Public BI data (authors)
FastLanes	CWI, PVLDB 2025	1024-value vectors decodable at any SIMD width, multi-column compression	Research prototype	Decoding 43x faster than Parquet with Snappy, 2% better compression than Parquet with Zstd (authors)
Lance	LanceDB, 2022	Random access for AI data, no row groups, vectors and blobs	File versions 2.0 to 2.2 stable, 2.3 unstable	Design target of 1 I/O per fixed-width lookup, 2 per variable-width (authors)
Nimble	Meta, 2023	Thousands of feature columns, FlatBuffers metadata	Work in progress, no versioning guarantees	No public benchmark found
Vortex	SpiralDB, now LF AI & Data	Extensible encodings, compute on compressed data, zero-copy Arrow	File format stable since release 0.36.0; Incubation stage since August 2025	100x random access, 10 to 20x scans, 5x writes vs Parquet (vendor claim, see next section)
F3	CMU, Tsinghua and others, PACMMOD 2025	WebAssembly decoders embedded in every file	Research prototype, not for production	Decoders cost kilobytes per file (authors)

Lightweight cascades: BtrBlocks, FastLanes and ALP

BtrBlocks, FastLanes and ALP share one idea: drop general-purpose compression and stack cheap encodings (run-length, dictionary, frame of reference with bit-packing) chosen per block from a sample of the data. Decoding then runs at memory speed. FastLanes lays out values so that the same bytes decode in parallel at any SIMD width, and its paper reports decoding 43 times faster than Parquet with Snappy while compressing 2 percent better than Parquet with Zstd.

Read the FastLanes random-access figure with care. The 315 to 416 times speedup over Parquet in the paper was measured by fetching the first row of a file (`LIMIT 1 OFFSET 0`), which tests time to first row, not scattered point lookups. ALP applies the cascade idea to floating-point data: most real doubles are decimals in disguise, so ALP scales them to integers, verifies the round trip, and bit-packs the result. It is DuckDB's default float compression.

Lance: random access for AI datasets

Lance removed row groups on purpose. Its specification states the position plainly:

We believe the concept of row groups to be fundamentally harmful to performance.

Lance file format specification, lance.org

Each Lance column has its own pages. Small values go into mini-blocks of 4 to 8 KiB, and large values such as embeddings or images use a per-value addressable layout, which is how the format reaches one

or two I/O operations per lookup. Lance also brings a dataset layer with versioning and vector indexes (IVF-PQ, HNSW). The trade is size: on Vortex's compression benchmark corpus, Lance files average about twice the size of Parquet with Zstd, a figure published by a competitor.

Nimble: a library rather than a specification

Meta built Nimble, formerly Alpha, for ML feature tables with thousands of columns, replacing Thrift metadata with FlatBuffers and making encodings pluggable. Its README is explicit about the governance model:

More than a specification, Nimble is a product. We strongly discourage developers to implement Nimble's spec.

Nimble README, github.com/facebookincubator/nimble

Chris Riccomini framed the consequence in 2024: extensible encodings cut both ways, because a reader that meets an encoding it does not know cannot decode the file. Parquet's strength is dozens of independent readers. Nimble trades that for speed of evolution inside the Velox stack, and its README still warns of no stability or versioning guarantees.

Vortex: the general-purpose candidate

Vortex started at SpiralDB in 2024, was donated to the LF AI & Data Foundation in June 2025, and reached the Incubation stage in August 2025, with support announced from Microsoft, Snowflake and Palantir. It separates the logical schema from the physical encodings, runs compute kernels directly on compressed arrays, and hands data to engines as zero-copy Arrow. The file format has been backward compatible since release 0.36.0, with integrations for DuckDB, DataFusion and Spark.

F3: shipping the decoder with the data

F3, from the team behind the 2023 evaluation (Zeng, McKinney, Pavlo, Zhang and co-authors, PACMMOD September 2025), answers the compatibility problem directly: each file embeds WebAssembly binaries that can decode its data, at a cost of kilobytes per file, so a reader without a native decoder still works. TUM's AnyBlox (PVLDB 2025) takes the same approach. Both are research prototypes, but they target the problem that slows every new encoding: readers lag writers.

Published speedups depend on the storage tier

The same format can be 56 times faster or about 2 times faster than Parquet depending on where the file sits. Vortex's own benchmark dashboard, read on 21 October 2026, shows hot random access on local NVMe about 56 times faster than Parquet, but only 2.1 to 2.5 times faster on S3, where request latency dominates decode time. A headline speedup without the storage tier is not a usable number.

Metric	Vortex launch claim (August 2025)	Vortex dashboard, 21 October 2026
Random access vs Parquet	100x faster	About 56x on NVMe hot cache, 37x cold, 2.1 to 2.5x on S3
Scan throughput vs Parquet	10 to 20x faster	5.57x geometric mean
Write throughput vs Parquet	5x faster	0.99x geometric mean
Compressed size	Similar to Parquet	1.07x Parquet with Zstd on average
Fastest on TPC-H over S3	n/a	DataFusion reading Parquet
Fastest on TPC-H over NVMe	n/a	DuckDB native format

This is to Vortex's credit: the project publishes the dashboard that qualifies its own headline, and its numbers change with every commit. Blog summaries drift faster than dashboards, and several articles still repeat older README figures of 100 to 200 times. Treat any figure older than the last release as a pointer, not as a fact.

I read every file format benchmark with three questions: which storage tier and cache state, which baseline writer settings, and which engine reads each format. A Parquet baseline written with defaults, read cold from S3 by one engine and hot from NVMe by another, can produce almost any ratio. The performance and cost engineering guide applies the same hygiene to engine and hardware claims.

GeoParquet is becoming plain Parquet

GeoParquet standardized vector geometries in Parquet: WKB geometry columns plus file-level metadata for the coordinate reference system and bounding boxes (version 1.0, September 2023). Parquet then absorbed the idea. Native GEOMETRY and GEOGRAPHY types with per-column-chunk bounding-box statistics shipped in parquet-format 2.11.0 (March 2025), and GeoParquet 2.0.0-rc.1 (July 2026) now requires them as the storage encoding.

GeoParquet 1.1 (June 2024) had added an optional bounding-box column and GeoArrow-based native encodings. The 2.0 release candidate drops both, on the grounds that Parquet's native row-group statistics do this job more efficiently and point files get smaller. GeoParquet is developed by an OGC Standards Working Group but is not yet an approved OGC standard.

The table layer followed. Apache Iceberg's v3 specification adds geometry and geography types, the Java implementation landed in Iceberg 1.9.0 (April 2025), and Snowflake made Iceberg v3 tables with both types generally available in May 2026. On the tooling side, GDAL 3.12 writes the native Parquet types when built against Arrow 21 or later.

Spatial order decides whether pruning works

GeoParquet has no spatial index. Spatial queries skip data through row-group bounding boxes, and those only help when the rows inside each group are close on the map. A file written in arrival order has row groups whose boxes each cover the whole area, so every query reads every row group. The GeoParquet community guide recommends spatial sorting (Hilbert curve, GeoHash, S2 or a KD-tree) and row groups of 50,000 to 150,000 rows.

Overture Maps applies this at scale: its buildings layer, more than 2.2 billion rows, is ordered by GeoHash with about 150,000 rows per row group, according to the same guide. DuckDB spatial and GDAL (`S0RT_BY_BB0X`) can sort at write time, and Iceberg added a Hilbert-curve clustering strategy for data file rewrites. Spatial statistics exist at row-group level only; page-level spatial statistics are still listed as an open exploration.

Cloud-optimized geo formats share one access pattern

Every cloud-optimized geospatial format follows the same rule: put a small index where one request can fetch it, then read only the byte ranges a query needs over HTTP. Cloud Optimized GeoTIFF, Zarr v3 with sharding, PMTiles, FlatGeobuf and COPC differ in data model, and each applies that rule to imagery, arrays, map tiles, vector features or point clouds stored on object storage.

Format	Data	Index read first	Status
Cloud Optimized GeoTIFF (COG)	Raster imagery	Tiled image directories plus overviews; HTTP range reads required	OGC standard 21-026, approved May 2023

Format	Data	Index read first	Status
Zarr v3 with sharding	N-dimensional arrays (climate, Earth observation cubes)	Shard index at the start or end of each object	Core spec 2023, sharding November 2023, zarr-python 3.0 in January 2025
PMTiles v3	Vector or raster map tiles	Header and root directory within the first 16 KiB; tiles ordered on a Hilbert curve	Single-file archive by Protomaps
FlatGeobuf	Vector features	Optional packed Hilbert R-tree after the header	Supported in GDAL, QGIS, PostGIS, GeoServer
COPC	LIDAR point clouds	Clustered octree inside a LAZ 1.4 file; 589 bytes identify it	COPC 1.0 specification; the file is a LAZ 1.4 file
MapLibre Tile (MLT)	Vector map tiles	Column-oriented tile with lightweight encodings	Announced January 2026 as the successor to Mapbox Vector Tiles

Zarr's sharding codec states the storage reason in its specification: cloud object stores such as S3 and GCS do not handle large numbers of small objects efficiently, while array chunks must stay small for reads. Sharding packs many chunks into one object with a byte-range index. The same pressure explains PMTiles, whose documentation contrasts it with MBTiles: one file read remotely with at most two cacheable intermediate requests, instead of a SQLite database behind a tile server.



Cloud-optimized formats fetch a small index first, then only the byte ranges a query needs. Request count, not file size, sets the latency.

The FlatGeobuf documentation names the limiting factor: round-trip latency more often than throughput. On satellite imagery and geospatial pipelines, the format question is usually settled early; the performance question is chunk, tile and shard size against request latency on the storage you actually have. I size these workloads on requests per query first and bytes second. MLT is the newest entry, with project-published claims of up to 6 times better compression on large tiles and SIMD-friendly decoding.

For LIDAR, the compression layer matters as much as the index. LAZ, produced by the LASzip library that Martin Isenburg presented in 2011, compresses LAS point clouds losslessly to 7 to 20 percent of their original size according to its maintainers, and decompresses to a bit-identical LAS file. It became the de facto standard for compressed LIDAR. A COPC file is itself a LAZ 1.4 file reorganized as a clustered octree, limited to LAS point formats 6, 7 and 8.

In my practice, plain LAZ is the better default: it is what archives, exchanges and full-file processing pipelines handle best, with the widest tool support. COPC earns its place when clients read spatial subsets of large point clouds remotely over HTTP.

Time-series data: specialized encodings, generic containers

Time-series storage split in two. The encodings stayed specialized: Facebook's Gorilla paper (VLDB 2015) compressed monitoring data to an average of 1.37 bytes per point, a 12 times reduction, with delta-of-delta timestamps and XOR-encoded floats, and Prometheus still uses that scheme. The containers went generic: InfluxDB 3 was rebuilt on Arrow, DataFusion and Parquet, and GreptimeDB and QuestDB write Parquet files too.

Gorilla works because monitoring data is regular. On Facebook's production data, about 96 percent of timestamps compressed to a single bit, and about 51 percent of values were identical to the previous one and also took one bit. Prometheus groups samples into two-hour blocks, compacts them up to 10 percent of retention or 31 days, and documents 1 to 2 bytes per sample. Thanos and Grafana Mimir store those same blocks in object storage.

The format still evolves, with the usual compatibility cost. Prometheus 3.15.0 (September 2026) stabilized a new XOR2 chunk encoding and warns that any other software reading TSDB blocks directly must support it. InfluxData reports that Parquet compressed its data 5 times better than specialized time-series formats once index structures are counted, measured on a non-public subset of its cloud data. That is a vendor figure without a public method.

Float compression after Gorilla

Three peer-reviewed schemes improved on Gorilla's XOR encoding for floats, each measured by its own authors on public datasets. ALP is the one that spread: DuckDB replaced Chimp and Patas with it in version 0.10.0 (February 2024), ClickHouse lists it as a beta column codec, and Parquet 2.14.0 adopted it in Preview. One encoding now sits in a database engine, a time-series codec list and the Parquet spec.

Scheme	Venue	Published result (authors' measurements)
Chimp, Chimp128	PVLDB 2022	About half the space of earlier streaming float schemes
Elf	PVLDB 2023	Compression ratio 43.9% better than Gorilla, 12.4% better than Chimp128
ALP	SIGMOD 2024	Compression ratio about 49% better than Gorilla and 24% better than Chimp128, with compression and decompression one to two orders of magnitude faster

Your table format decides what you can deploy

Most lakehouse data sits under a table format, and that layer limits the file format choice. The Apache Iceberg specification still lists only Avro, ORC and Parquet as data file formats. Iceberg 1.11.0 (May 2026) shipped a pluggable File Format API, a pull request adding Vortex is open, and the Lance pull requests were closed unmerged, as of October 2026.

Parquet and Iceberg evolve together: the VARIANT and geospatial types arrived in Parquet 2.11.0 alongside Iceberg v3, which also added binary deletion vectors. Lance brings its own dataset layer with versions and indexes, so in practice it acts as a separate table format. Alex Merced, writing for Dremio, describes the pattern that follows: Parquet under Iceberg as the governed source of truth, with a Lance dataset derived from it for retrieval and training.

How to choose a file format by access pattern

Choose from the access pattern first, then the storage tier, then the list of readers. Batch scans on object storage still favor Parquet with good writer settings. Point lookups on embeddings and blobs favor Lance. Scan-heavy analytics on local NVMe is where Vortex and DuckDB's native format lead. Geospatial data follows the cloud-optimized format of its data model, and time series stays in its engine's native blocks.

Workload	Default choice	Worth testing	Check first
Streaming records, Kafka payloads	Avro	n/a	Schema evolution rules
Batch analytics on object storage	Parquet with Zstd, sized row groups, page index	Vortex, once your engines read it	Writer settings, requests per query
Interactive analytics on local NVMe	Parquet or the engine's native format	Vortex	Decode CPU against I/O wait
Random access, embeddings, multimodal data	Lance	Parquet tuned for random access	I/O operations per lookup on your tier
Very wide ML feature tables	Parquet with minimal statistics	Nimble inside a Velox stack	Footer parse time per file
Vector geometries	Parquet native geo types (GeoParquet 2.0), spatially sorted	FlatGeobuf for web streaming	Overlap of row-group bounding boxes
Imagery and arrays	COG; Zarr v3 with sharding	VirtualiZarr over existing NetCDF or HDF5	Chunk size against request latency
Map tiles	PMTiles carrying MVT tiles	MLT	Client support
LIDAR point clouds	LAZ (LASzip)	COPC for remote spatial subset reads	Whether clients read whole files or spatial subsets
Metrics and monitoring	The TSDB's native blocks	Parquet export for long-term analytics	Reader support for new chunk encodings

For wide feature tables, InfluxData's 2024 measurements on files with up to 100,000 columns found that omitting column statistics made metadata decode about 30 percent faster and smaller, and that Parquet metadata overhead is smaller than generally recognized. Settings come before a format change.

Before any format migration, I run the same sequence:

1. Rewrite a sample with tuned Parquet settings and measure it. This is the real baseline.

2. Benchmark the candidate on your storage tier, in your cache state, with your queries and your engine.
3. List every reader: query engines, the table format, BI tools, ML data loaders, and backup or replication tools that parse files.
4. On object storage, count requests per query and per lookup, then bytes.
5. Keep an interchange copy in Parquet and a rollback path until every reader supports the new format.

When a format change is about to drive a hardware purchase or a platform migration, a Flash Architecture Review settles that one decision on measured evidence. Capacity planning and platform sizing translates the result into drives, servers and object storage requests. The 15-minute intro call is free.

FAQ

Is Parquet obsolete?

No. Parquet remains the most widely readable columnar format and keeps adding features: VARIANT and geospatial types in format 2.11.0 (2025) and the ALP float encoding in 2.14.0 (2026). Its weak spots are point lookups, very wide footers and decode cost on fast local storage. Tuned settings recover much of the gap; the Lance authors measured random access more than 60 times faster with tuned parquet-rs than with defaults.

What is the difference between GeoParquet and Parquet geometry types?

GeoParquet 1.x is a metadata convention on top of Parquet: WKB geometry columns plus file-level metadata for the coordinate reference system and bounding box. Parquet format 2.11.0 (2025) added native GEOMETRY and GEOGRAPHY logical types with per-column-chunk bounding-box statistics. GeoParquet 2.0.0-rc.1 (July 2026) requires those native types, so a GeoParquet 2.0 file is a Parquet file using them, with optional extra metadata.

Is Lance better than Parquet for machine learning?

Lance is better for random access to individual rows, embeddings and large binary values, which dominate retrieval and training-sample loading. Its design targets one or two I/O operations per lookup and it adds versioning and vector indexes. Parquet remains stronger for compression and broad engine support: on Vortex's benchmark corpus, Lance files average about twice the size of Parquet with Zstd. Many teams keep both.

Can Apache Iceberg use Lance or Vortex files?

Not in a released version as of October 2026. The Apache Iceberg specification lists Avro, ORC and Parquet as data file formats. Iceberg 1.11.0 (May 2026) added a pluggable File Format API that makes other formats possible, a pull request adding Vortex is open, and the pull requests for Lance were closed without being merged.

Which compression is best for time-series floating-point data?

ALP (SIGMOD 2024) currently leads the published comparisons: its authors report a compression ratio about 49 percent better than Gorilla and 24 percent better than Chimp128, with one to two orders of magnitude faster compression and decompression. DuckDB uses it by default since version 0.10.0, ClickHouse offers it as a beta codec, and Parquet 2.14.0 includes it in Preview. Check that every reader supports it first.

Sources

- Zeng, Hui, Shen, Pavlo, McKinney, Zhang, An Empirical Evaluation of Columnar Storage Formats, PVLDB 17(2), 2023
- Pace et al., Lance: Efficient Random Access in Columnar Storage through Adaptive Structural Encodings, arXiv, April 2025; Lance file format specification
- Kuschewski, Sauerwein, Alhomssi, Leis, BtrBlocks: Efficient Columnar Compression for Data Lakes, SIGMOD 2023
- Afroozeh, Boncz, The FastLanes File Format, PVLDB 18(11), 2025
- Afroozeh, Kuffó, Boncz, ALP: Adaptive Lossless floating-Point Compression, SIGMOD 2024
- Zeng et al., F3: The Open-Source Data File Format for the Future, PACMMOD 3(4), 2025; Gienieczko et al., AnyBlox, PVLDB 18(11), 2025
- Vortex at LF AI & Data; Vortex benchmark dashboard, read 21 October 2026
- Nimble README, Meta
- apache/parquet-format releases; Parquet implementation status; Parquet geospatial types
- Kuiper, Parquet encodings in DuckDB, January 2025; 3x to 9x faster Parquet footer metadata in Rust, Apache Arrow, October 2025; Hao and Lamb, How good is Parquet for wide tables, InfluxData, June 2024
- GeoParquet releases; Distributing GeoParquet; Apache Iceberg specification; Iceberg File Format API
- OGC Cloud Optimized GeoTIFF 21-026; Zarr v3 sharding codec; PMTiles v3 specification; FlatGeobuf; LASzip; COPC; MapLibre Tile announcement
- Pelkonen et al., Gorilla: A Fast, Scalable, In-Memory Time Series Database, VLDB 2015; Prometheus chunk format; The FDAP architecture behind InfluxDB 3.0, InfluxData
- Liakos et al., Chimp, PVLDB 2022; Li et al., Elf, PVLDB 2023
- Further reading: Dipankar Mazumdar, Apache Parquet vs. Newer File Formats (2025); Alex Merced, The File Format Renaissance (2026) and Exploring the Evolving File Format Landscape, Dremio (2025); Chris Riccomini, Nimble and Lance: The Parquet Killers, Materialized View (2024)

Benchmark figures are attributed to whoever measured them; vendor figures are labeled as such. I have no affiliation with any project or vendor cited here, and none of them reviewed or endorsed this article.

Related posts

- Performance and cost engineering for data platforms
- GPUs changed everything: storage is the bottleneck again
- Sail: when Apache Spark meets Rust, a practitioner's deep dive
- Benchmark best practices, part 1: Amdahl's law caps your scaling claim

PDF snapshot of the public article. The online version is authoritative; links remain clickable.