Format-Aware Compression for Telemetry Data Transport: Can Format-Aware Compression Match Native Columnar Protocols?

Danny Chin
Carnegie Mellon University
Pittsburgh, PA, USA

Pragna Mamidipaka
Carnegie Mellon University
Pittsburgh, PA, USA

ABSTRACT

No—at least not yet. We evaluate OpenZL, a schema-driven compression framework, against OTel Arrow (OTAP) across five workloads and find it achieves only 45–91% of OTAP's compression ratio for telemetry data. Through an ablation study, we identify OTAP's sorting optimization as the key differentiator: when disabled, OpenZL outperforms OTAP in three of four workloads. However, for TPC-H LineItem, a table with flat schema, OpenZL outperforms Arrow in our evaluation, suggesting the gap may stem from OTAP's domain-specific optimizations rather than inherent format limitations.

1 INTRODUCTION

Modern distributed systems generate telemetry data (metrics, traces, and logs) at unprecedented scales. A typical large-scale enterprise deployment may produce terabytes of observability data daily, consuming significant network bandwidth and storage resources. The OpenTelemetry (OTel) project has emerged as the industry standard for telemetry data collection, initially using the OpenTelemetry Protocol (OTLP), a row-oriented protocol based on Protocol Buffers and gRPC.

To address the growing data volumes, the OpenTelemetry community developed OTel Arrow (OTAP), a columnar transport protocol built on Apache Arrow that achieves significantly better compression ratios than OTLP. However, adopting OTAP requires substantial engineering effort: protocol stack changes, client-server compatibility concerns, and modifications to existing infrastructure. This raises a fundamental question: *Is protocol migration necessary, or can sophisticated compression alone bridge the efficiency gap?*

This paper investigates whether OpenZL, a format-aware compression framework that automatically generates schema-specific compression plans, can achieve compression performance comparable to native columnar protocols when applied to row-oriented data. If successful, such an approach would allow organizations to retain their existing row-oriented infrastructure while achieving near-columnar compression efficiency, which is a significant practical advantage.

Our key contributions are:

- A comprehensive evaluation of format-aware compression (OpenZL) on row-oriented transport protocol versus standard compression on native columnar protocols (OTAP) across five workloads.
- An ablation study decomposing OTAP's compression advantage into its constituent optimizations (sorting, delta dictionary, columnar deduplication).
- Practical guidance on when format-aware compression can substitute for protocol migration versus when columnar protocols remain essential.

Our evaluation framework, including benchmark tools and data generation scripts, is open source [3].

2 MOTIVATION

2.1 The Scale of Telemetry Data

Large-scale distributed systems produce massive volumes of telemetry data. Consider a typical cloud provider managing 100,000 servers, each emitting metrics every 10 seconds. This generates 10,000 metric data points per second per server, totaling 1 billion data points per second fleet-wide. At an average of 100 bytes per uncompressed data point, this represents 100 GB/s of raw telemetry traffic.

Even modest compression improvements translate to substantial resource savings. A 10% improvement in compression ratio could save network bandwidth of 10 GB/s. It could save storage costs at $0.02/GB per month, i.e., approximately $200,000 per month. It will also save CPU resources by reduced decompression overhead on query nodes.

2.2 The Protocol Migration Challenge

The OpenTelemetry community's decision to develop OTAP reflects the recognized need for better compression. However, protocol migration introduces significant challenges:

1. **Infrastructure changes:** Modifications required across collectors, agents, and backend systems
2. **Compatibility management:** Supporting both protocols during transition periods
3. **Engineering effort:** Thousands of engineering hours for large deployments

2.3 Format-Aware Compression: A Third Way?

Format-aware compression frameworks like OpenZL offer a potential alternative. By learning the structure and patterns within protocol schemas, they can exploit field-level redundancy without data reorganization, apply specialized compression techniques based on data types, and adapt to workload-specific patterns through training.

If format-aware compression can approach the efficiency of columnar protocols, it would provide a compelling alternative: retain existing infrastructure while achieving near-columnar compression benefits. However, this remains an open empirical question that our work addresses.

3 BACKGROUND

3.1 OpenTelemetry Protocol (OTLP)

OTLP is the standard row-oriented protocol for transmitting OpenTelemetry data. It uses Protocol Buffers (protobuf) for serialization and gRPC for transport.

3.1.1 Protocol Buffers and gRPC. Protocol Buffers is a language-neutral, platform-neutral mechanism for serializing structured data. It defines schemas in a domain-specific language that specifies message types, fields, and data types. The protobuf compiler generates code for serializing and deserializing messages in various programming languages. gRPC is a high-performance RPC framework that uses HTTP/2 for transport and Protocol Buffers as its interface definition language.

3.1.2 OTLP Data Structure. OTLP organizes telemetry data hierarchically. For metrics:

```
ExportMetricsServiceRequest
|-- ResourceMetrics[]
    |-- Resource (attributes)
    |-- ScopeMetrics[]
        |-- Scope (name, version)
        |-- Metric[]
            |-- Name, Description, Unit
            |-- MetricType (Gauge/Sum/...)
            +-- DataPoints[]
                |-- Timestamp
                |-- Value
                +-- Attributes[]
```

For traces:

```
ExportTraceServiceRequest
|-- ResourceSpans[]
    |-- Resource (attributes)
    |-- ScopeSpans[]
        |-- Scope (name, version)
        |-- Span[]
            |-- TraceId, SpanId
            |-- ParentSpanId
            |-- Name, Kind
            |-- StartTime, EndTime
            |-- Attributes[]
            |-- Events[]
            +-- Links[]
```

The batch size in our experiments refers to the number of leaf-level data points in a single payload: the number of metric DataPoints for metrics workloads, or the number of Spans for trace workloads, and the number of rows for TPC-H.

3.1.3 OTLP Compression Pipeline. The standard OTLP compression pipeline consists of:

(1) **Schema Definition:** Protobuf schema defines message structure, field types, and encoding rules (e.g., endianness, varint encoding)
(2) **Serialization:** Data structures are serialized into binary protobuf format following the schema
(3) **Compression:** The serialized bytes are compressed using zstd (or OpenZL in our experiments)
(4) **Transport:** Compressed payload is transmitted over gRPC

3.2 OTel Arrow (OTAP)

OTel Arrow represents a fundamental redesign of the transport protocol using columnar organization. This section describes both the data format and three key optimizations that contribute to OTAP's compression efficiency: delta dictionary, sorting, and columnar deduplication.

3.2.1 Apache Arrow and IPC Format. Apache Arrow is an in-memory columnar data format designed for efficient analytics. The Arrow IPC (Inter-Process Communication) format extends Arrow to support streaming and serialization. An Arrow IPC stream consists of:

```
Arrow IPC Stream
    |-- Schema Message
    |    +-- Column definitions (types, names)
    |-- Dictionary Messages (optional)
    |    +-- Shared value mappings
    +-- RecordBatch Messages
        |-- Batch metadata (row count)
        +-- Column buffers (actual data)
```

Instead of storing data row-by-row, Arrow organizes data in columns, enabling:

- **Better cache locality:** Accessing a single field requires reading only one column
- **Compression efficiency:** Similar values grouped together compress better

3.2.2 *Data Organization.*

OTAP transforms OTLP's nested row-oriented structure into multiple related columnar tables (star schema). For metrics:

```
Metrics Table (main):
|- id, resource, scope, metric_type
|- name, description, unit (dict encoded)
+- aggregation_temporality, is_monotonic

DataPoints Table (related):
|- parent_id -> Metrics.id
|- timestamp, value, flags
+- attributes (dict refs)
```

Each column stores values for a single field. String columns use dictionary encoding, replacing repeated values with compact integer references.

3.2.3 *Optimization: Delta Dictionary.*

OTAP uses delta dictionary encoding to efficiently transmit string values across batches. Instead of sending complete dictionaries with each batch, OTAP maintains a stateful dictionary across the connection: only new strings are transmitted as deltas. For example, if the first batch contains metric names `["cpu", "mem"]`, these are assigned IDs 0 and 1. When a subsequent batch uses the same metric names, it references IDs 0 and 1 without retransmitting the strings.

3.2.4 *Optimization: Sorting.*

OTAP sorts rows based on column values to improve compression. Consider unsorted data:

```
resource: [B, A, B, A]
metric: [cpu, mem, mem, cpu]
```

After sorting by resource, then metric:

```
resource: [A, A, B, B]
metric: [cpu, mem, cpu, mem]
```

The sorted columns have longer runs of identical values, enabling more effective run-length encoding and delta compression.

Sorting is straightforward in columnar formats because data is already flattened into tables. In OTLP's nested structure (Resource $\rightarrow$ Scope $\rightarrow$ Metric $\rightarrow$ DataPoints), reordering data points would break hierarchical groupings or require duplicating parent metadata.

3.2.5 *Optimization: Columnar Deduplication.*

OTAP deduplicates resource and scope metadata across consecutive data points. When a new data point arrives, OTAP checks if its resource/scope matches the previous one. If so, it reuses the existing reference; otherwise, it stores new metadata. This reduces redundancy for workloads where many consecutive data points share common resources.

3.3 OpenZL: Format-Aware Compression

OpenZL is a general-purpose compression framework that learns format-specific compression strategies through automated training.

3.3.1 *Core Concept.*

Unlike general-purpose compressors like zstd and gzip that treat data as opaque byte streams, OpenZL parses structured data to understand field boundaries, types, and relationships. It then automatically generates field-specific compression strategies and compiles optimized compression code tailored to the schema.

3.3.2 *Training Process.*

OpenZL's training process consists of:

(1) **Parser Implementation:** Developers implement a parser that extracts structured data from the serialized format. For OTLP, this involves parsing protobuf messages according to the OTel schema.
(2) **Training Data Collection:** Sample payloads from the target workload are collected. These samples must be representative and are strictly separated from evaluation data to prevent overfitting.
(3) **Training:** OpenZL analyzes the training samples and generates encoding strategies for each field, producing a trained compressor file.

3.3.3 *OpenZL Integration for OTLP.*

To apply OpenZL to OTLP, we implemented a protobuf parser for OTLP metrics and traces, registered it with OpenZL's training framework, trained separate compressors for metrics and traces workloads, and integrated the trained compressors into the compression pipeline. The key distinction from standard compression approaches is that OpenZL operates on structured protobuf data rather than raw byte streams, enabling it to exploit semantic patterns that are invisible to byte-level compressors.

3.4 Compression Metrics

We evaluate compression systems using three primary metrics:

- **Compression Ratio:** $CR = \frac{\text{Original Size}}{\text{Compressed Size}}$. Higher is better. We also report bytes per data point for interpretability.
- **Compression Speed:** Throughput of compression in MB/s. Critical for high-throughput telemetry systems where compression occurs on the hot path.
- **Decompression Speed:** Throughput of decompression in MB/s. Important for query-time performance.

4 RESEARCH GOALS AND HYPOTHESES

4.1 Primary Goal

Determine whether format-aware compression (OpenZL) applied to row-oriented protocols (OTLP) can achieve compression performance comparable to native columnar protocols (OTAP).

4.2 Experimental Hypotheses

We test three specific hypotheses:

(H1) OpenZL improves row-oriented compression: OpenZL should achieve better compression ratios than standard zstd when applied to OTLP. Expectation: OTLP+OpenZL compression ratio > OTLP+zstd compression ratio.

(H2) Columnar format aids compression: Even with identical compression algorithms, columnar organization should improve compression ratios. Expectation: OTAP+zstd compression ratio > OTLP+zstd compression ratio.

(H3) Format-aware compression bridges the gap: Most critically, format-aware compression should enable row-oriented protocols to approach columnar protocol efficiency. Expectation: OTLP+OpenZL compression ratio ≈ OTAP+zstd compression ratio.

If H3 holds, it would suggest that organizations can achieve near-columnar compression without protocol migration by adopting format-aware compression.

4.3 Secondary Goals

Beyond compression ratios, we investigate:

- **Compression and decompression speeds:** Whether OpenZL's training-based approach introduces prohibitive compression or decompression latency
- **Batch size sensitivity:** How compression performance varies with payload size (critical for understanding deployment scenarios)
- **Workload generalization:** Whether OpenZL's benefits extend to other structured data workloads beyond telemetry. To evaluate this, we repeat all experiments on the TPC-H benchmark, a standard database workload with different data characteristics.
- **Understanding OTAP's advantage:** What specific optimizations in OTAP contribute most to its compression efficiency, and under what conditions can format-aware compression match or exceed OTAP variants with specific optimizations disabled.

5 METHODOLOGY

5.1 Experimental Setup

5.1.1 Protocol Implementations. We evaluate three protocol-compressor combinations:

(1) **OTLP + zstd:** Baseline row-oriented protocol with standard compression
(2) **OTLP + OpenZL:** Row-oriented protocol with format-aware compression
(3) **OTAP + zstd:** Columnar protocol with standard compression

For OTAP, we additionally evaluate ablation variants with individual optimizations disabled (sorting, delta dictionary, columnar deduplication, and dictionary encoding).

5.1.2 Batch Size Variation. To understand how compression scales with payload size, we vary batch sizes from 10 to 50000 data points.

5.2 Workload Selection

We selected four OpenTelemetry workloads, and a TPC-H benchmark to ensure generalizability.

(1) **Astronomy Metrics:** Metrics from astronomical observation systems. Characterized by periodic sensor readings with stable schemas.
(2) **Astronomy Traces:** Distributed traces from astronomy data processing pipelines. Features long-running spans with complex attribute sets.
(3) **Hipstershop Metrics:** Metrics from the OpenTelemetry demo application (a microservices e-commerce system). Represents typical cloud-native application metrics.
(4) **Hipstershop Traces:** Traces from the same demo application.
(5) **TPC-H LineItem:** The largest relation in the TPC-H schema, representing granular details of individual orders. Unlike the nested hierarchical structure of OTel data, TPC-H has a flat schema with no nested messages. Included to test generalizability beyond telemetry data.

5.3 Data Collection and Transformation

5.3.1 OTLP Data Preparation. For each workload, we sourced raw telemetry data in OTLP format and split it into training (10%) and test (90%) sets using random sampling at the data point level. This ensures proper separation between training and evaluation data while maintaining representative samples from each workload. For testing, we generated payloads of varying batch sizes (10 to 50,000 data points) and serialized them to protobuf format.

5.3.2 Data Generation Tools. We developed two tools for generating test payloads (available at [3]). The rebatch tool (for OTel workloads) reads raw telemetry data from zstd-compressed OTLP files, rebatches data points into specified batch sizes, and outputs in multiple formats. For OTLP output, it serializes using the standard protobuf marshaler. For

OTAP output, it uses the official otel-arrow library's Producer, which converts OTLP data structures to Arrow columnar format with dictionary encoding, sorting, and columnar deduplication. The tool also supports ablation variants that disable individual optimizations.

The tpch-gen tool (for TPC-H workload) reads TPC-H .tbl files (pipe-delimited CSV) and generates batches in both protobuf and Arrow IPC formats. For Arrow output, it builds records with dictionary encoding on low-cardinality string columns and supports incremental dictionary deltas across batches.

5.4 OpenZL Training

For each workload, we implemented OTLP parser variants for metrics and traces, and trained OpenZL compressors on the training data. Training payloads were generated at batch size 1000 from the 10% training split, providing sufficient data point diversity while maintaining reasonable payload sizes. Training was performed once per gRPC schema and the resulting compressors were reused across all batch sizes during evaluation.

5.5 Evaluation Procedure

For each combination of:

- Workload (5 options)
- Protocol (OTLP vs. OTAP)
- Compressor (zstd, OpenZL)
- Batch size (8 values from 10 to 50000)

We measured comporession ratios, compression speed and decompression speed.

6 EVALUATION RESULTS

The analysis in this section is based on the compression performance metrics presented in Table 1 and the batch size variations shown in Figures 1, 2, and 3.

6.1 Overall Compression Performance

We evaluate our three hypotheses based on the compression performance metrics presented in Table 1 and Figure 1:

(H1) OpenZL improves row-oriented compression: Supported. As shown in Figure 1, OpenZL (solid red line) mostly achieves higher compression ratios than zstd (dotted green line) for OTLP across all workloads, especially at larger batch sizes.

(H2) Columnar format aids compression: Strongly supported. Figure 1 shows OTAP+zstd (dotted blue line) consistently outperforms OTLP+zstd and OTLP+OpenZL, validating the OpenTelemetry community's motivation for developing OTAP. The columnar organization provides 1.3–2.2x improvement in compression over row-oriented format with the same compressor.

(H3) Format-aware compression bridges the gap: Not well supported for OTel workloads, but supported for TPC-H. For telemetry workloads in Figure 1, OTLP+OpenZL fails to reach OTAP+zstd even at large batch sizes. However, for TPC-H, OpenZL outperforms Arrow at larger batch sizes. This difference may be attributed to TPC-H's flat schema: the nested hierarchical structure of OTel data enables domain-specific optimizations in OTAP (sorting, columnar deduplication) that provide significant compression benefits, while flat schemas like TPC-H cannot leverage these optimizations as effectively. This raises the question: which specific optimizations give OTAP its edge for OTel workloads?

6.2 Decomposing OTAP's Advantage

To understand why H3 fails for OTel workloads, we conducted an ablation study examining four key optimizations in OTAP: sorting, delta dictionary, columnar deduplication, and dictionary encoding. We created variants that disable each optimization individually while keeping others enabled.

The results reveal that dictionary encoding is essential—without it, OTAP performs 2–6x *worse* than OTLP+zstd. Sorting is also critical: disabling it causes OTAP to underperform OTLP+zstd in half of the workloads. Delta dictionary and columnar deduplication provide diminishing returns at large batch sizes (1–11% degradation).

When sorting is disabled, OTLP+OpenZL outperforms OTAP in three of four OTel workloads. This demonstrates that OTAP's sorting optimization creates compression patterns that row-oriented formats cannot easily replicate.

6.3 Compression and Decompression Speed

Figures 2 and 3 show compression and decompression speeds across workloads. For OTel workloads, OpenZL demonstrates suboptimal performance in both compression and decompression. Notably, performance degrades slightly with larger batch sizes. This is a counterintuitive behavior that suggests potential optimization opportunities within OpenZL. With TPC-H workloads, compression speed shows improvement, though decompression remains considerably slow. This decompression bottleneck may stem from OpenZL's use of a universal decompressor that operates independently of the format-specific compression algorithms applied to individual files.

7 RELATED WORK AND DISCUSSION

7.1 Columnar Data Formats

The benefits of columnar data organization have been well-established in database systems and analytical workloads.

Table 1: Compression Performance Across Workloads (Batch Size = 50,000)

Format	Uncompressed			Zstd Compressed			OpenZL Compressed		
	Bytes	Ratio	By/pt	Bytes	Ratio	By/pt	Bytes	Ratio	By/pt
Hipstershop Metrics									
OTLP (Proto)	33.37M	1.00	556.0	469.8K	71.03	7.8	468.4K	71.25	7.8
OTAP (Arrow)	5.14M	6.49	85.7	209.2K	159.51	3.5	–	–	–
Hipstershop Traces									
OTLP (Proto)	80.13M	1.00	1027.7	2.64M	30.39	33.8	2.51M	31.97	32.1
OTAP (Arrow)	7.96M	10.06	102.1	1.64M	48.86	21.0	–	–	–
Astronomy Metrics									
OTLP (Proto)	269.91M	1.00	380.3	6.08M	44.39	8.6	4.99M	54.13	7.0
OTAP (Arrow)	58.57M	4.61	82.5	3.10M	87.06	4.4	–	–	–
Astronomy Traces									
OTLP (Proto)	56.71M	1.00	979.5	2.72M	20.83	47.0	2.36M	24.07	40.7
OTAP (Arrow)	12.33M	4.60	213.0	2.14M	26.55	36.9	–	–	–
TPC-H LineItem									
Proto	76.12M	1.00	140.8	20.45M	3.72	37.8	15.54M	4.90	28.7
Arrow	73.80M	1.03	136.5	17.03M	4.47	31.5	–	–	–

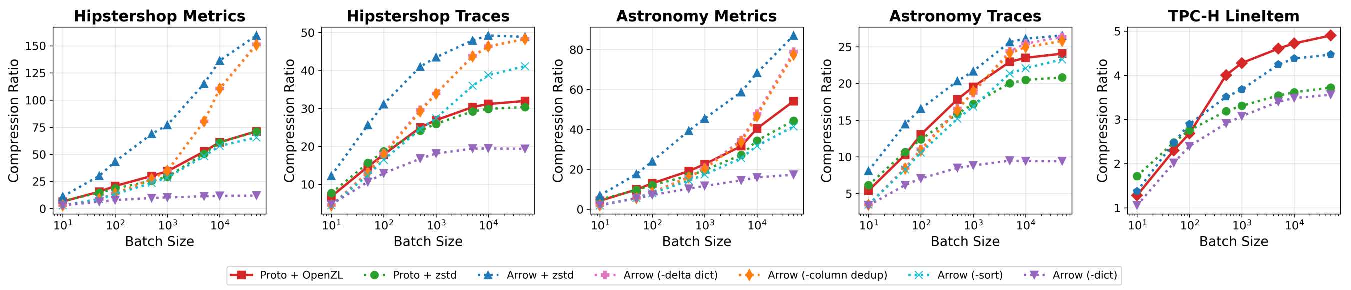

Figure 1: Compression ratio vs. batch size across all workloads. Higher is better. Arrow without sorting only outperforms Proto+OpenZL in Hipstershop Traces; OpenZL wins in the other three OTel workloads. For TPC-H, Proto+OpenZL performs similarly to Arrow.

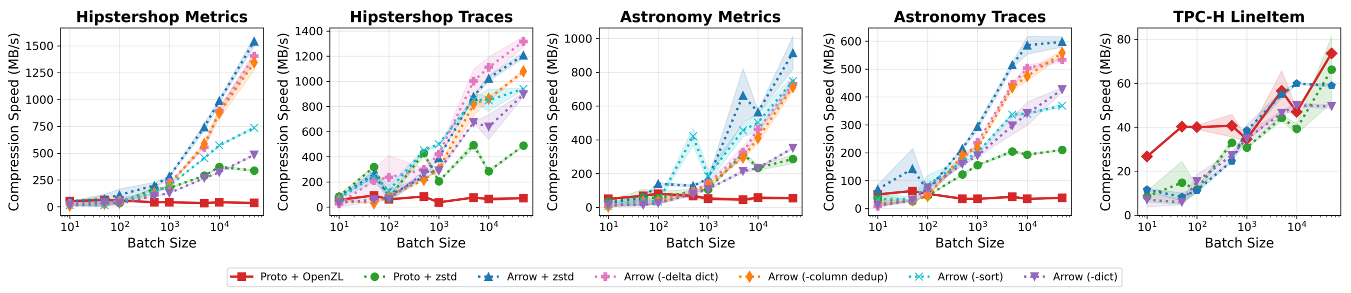

Figure 2: Compression speed vs. batch size across all workloads. Higher is better. OpenZL shows slower compression than zstd for OTel workloads but improves for TPC-H.

Apache Parquet [1] and ORC [8] demonstrated that columnar formats dramatically improve compression ratios and query performance for analytical workloads. Apache Arrow [6] extended these ideas to in-memory processing and interprocess communication.

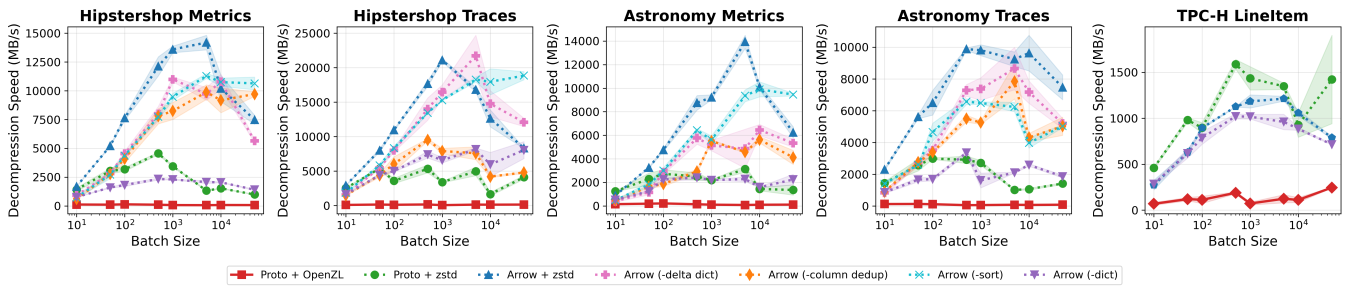

Figure 3: Decompression speed vs. batch size across all workloads. Higher is better. Unlike other compressors, OpenZL's decompression does not speed up with larger batch sizes, suggesting potential for performance improvement in OpenZL's implementation.

Table 2: Effect of Disabling Each OTAP Optimization (Batch Size 50,000)

Optimization Disabled	Effect
Dictionary Encoding	2–6x worse than OTLP+zstd
Sorting	Worse than OTLP+zstd in 2/4 workloads
Delta Dictionary	1–9% degradation from OTAP
Columnar Dedup	1–11% degradation from OTAP

OTel Arrow [2] represents the application of columnar principles to telemetry data transport. Our work complements this research by investigating whether format-aware compression can achieve similar benefits without requiring columnar reorganization.

7.2 Domain-Specific Compression

Specialized compression techniques have been developed for various domains:

Gorilla [9] introduced delta-of-delta encoding and XOR-based compression for time-series metrics, exploiting temporal locality and value similarity in metrics data. LogZip [5] uses pattern mining and neural networks to compress log data by identifying repeated templates and parameter values.

OpenZL extends these ideas to a general framework that can automatically discover and exploit domain-specific patterns through training, rather than requiring hand-crafted compression strategies.

7.3 Format-Aware Compression

Format-aware compression has been explored in several contexts. XMill [4] and XPRESS [7] separate structure from content in XML documents, enabling better compression of both components. Cap'n Proto's [10] "packing" compression removes sequences of zero bytes by encoding them efficiently with a tag byte system, achieving message sizes comparable to Protocol Buffers while remaining faster to process.

OpenZL differs from these approaches in its use of automated training to generate workload-specific compression plans, rather than relying on fixed heuristics. Our work set out to demonstrate that automated training on structured data can achieve compression performance competitive with manually-optimized protocols. However, we found that OTAP's performance remains difficult to surpass for OTel workloads due to its sorting optimization, which creates compression patterns that row-oriented formats cannot easily replicate. Despite this limitation, OpenZL serves as a viable alternative for non-OTel workloads where it performs comparably to columnar formats.

7.4 Practical Deployment Considerations

Based on our findings, we provide guidance on when each approach is appropriate, separated by workload type.

7.4.1 OTel Workloads. For telemetry data with deeply nested hierarchical schemas:

Use OTAP (columnar protocol) when maximum compression is the primary goal—OTAP achieves 1.3–2.2x better compression ratio than OTLP+zstd at all batch sizes, even as small as batch size 10.

Use OTLP+OpenZL (format-aware compression) when infrastructure changes are prohibitive or risky. For large batch sizes (1,000+ data points), OpenZL provides modest compression improvement over zstd while serving as a drop-in replacement.

Use OTLP+zstd (standard compression) when batch sizes are small (<1,000 data points) where OpenZL provides little to no compression improvement over zstd, or when simplicity and minimal dependencies are priorities.

7.4.2 Non-OTel Workloads. For structured data with flat schemas, our TPC-H evaluation shows that Proto+OpenZL outperforms Arrow+zstd at larger batch sizes. However, this is based on a single workload evaluation, and we recommend benchmarking on representative data before making deployment decisions.

For workloads with nested schemas that are not OTel-specific, performance may vary depending on schema complexity.

7.4.3 General Trade-offs. OpenZL requires training data collection and periodic retraining as workload characteristics evolve, adding operational complexity. Columnar protocols require format migration but provide predictable compression with standard zstd.

8 CONCLUSION

For telemetry data, format-aware compression cannot fully replace columnar protocols. We evaluated OpenZL against OTAP across five workloads and found that while OpenZL improves upon standard compression for row-oriented data, it achieves only 45–91% of OTAP's compression ratio for OTel workloads at large batch sizes.

Our ablation study revealed that OTAP's advantage stems primarily from dictionary encoding and sorting optimizations. Dictionary encoding is essential—without it, OTAP performs 2–6x worse than OTLP+zstd. When sorting is disabled, OTLP+OpenZL outperforms OTAP in three of four OTel workloads, demonstrating that sorting creates compression patterns that row-oriented formats cannot easily replicate due to their hierarchical structure.

For flat schemas like TPC-H, OpenZL outperforms Arrow in our evaluation, suggesting that format-aware compression may be a viable alternative when domain-specific optimizations are not in play. However, for telemetry data where OTAP has been extensively optimized, protocol migration remains the path to maximum compression efficiency.

REFERENCES

[1] [n. d.]. Apache Parquet. https://parquet.apache.org/. Accessed: 2024-12-03.

[2] [n. d.]. OpenTelemetry Arrow Protocol. https://github.com/open-telemetry/otel-arrow. Accessed: 2024-12-03.

[3] Danny Chin and Pragna Mamidipaka. 2024. openzl-row-col. https://github.com/danielthank/openzl-row-col. Benchmark code and data generation tools.

[4] Hartmut Liefke and Dan Suciu. 2000. XMill: An efficient compressor for XML data. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. 153–164.

[5] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, and Michael R Lyu. 2019. LogZip: Extracting Hidden Structures via Iterative Clustering for Log Compression. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. 863–873.

[6] Wes McKinney. 2016. Apache Arrow: A cross-language development platform for in-memory data. https://arrow.apache.org/.

[7] Jun-Ki Min, Myung-Jae Park, and Chin-Wan Chung. 2003. XPRESS: A queriable compression for XML data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. 122–133.

[8] Owen O'Malley et al. 2015. Major Technical Advancements in Apache Hive. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1235–1246.

[9] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. 2015. Gorilla: A fast, scalable, in-memory time series database. In *Proceedings of the VLDB Endowment*, Vol. 8. 1816–1827.

[10] Kenton Varda Sandstorm. 2013. Cap'n Proto: Introduction. https://capnproto.org/.