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

Danny Chin
Carnegie Mellon University
Pittsburgh, PA, USA

Pragna Mamidipaka
Carnegie Mellon University
Pittsburgh, PA, USA

ABSTRACT

High-volume telemetry systems face a fundamental trade-off between data transport efficiency and protocol complexity. While columnar protocols like OTel Arrow (OTAP) achieve superior compression through data reorganization, they require substantial protocol migration efforts. This paper investigates whether format-aware compression frameworks can obviate the need for such migrations by achieving comparable compression ratios on existing row-oriented protocols. We evaluate OpenZL, a general-purpose schema-driven compression framework, against the specialized OTAP protocol across five real-world workloads with varying batch sizes. Our results demonstrate that OpenZL applied to row-oriented OTLP achieves compression ratios around 20-50% of OTAP for large batches (1000+ data points). However, OpenZL exhibits significant computational overhead at small batch sizes, making traditional zstd more practical for such scenarios. These findings suggest that format-aware compression can serve as a viable alternative to protocol migration for batch-oriented telemetry systems, while highlighting the continued importance of columnar formats for maximum efficiency.

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 oreinted transport protocol versus standard compression on native columnar protocols (OTAP) across five workloads.
- Detailed analysis of the performance trade-offs across varying batch sizes.
- Practical guidance on when format-aware compression can substitute for protocol migration versus when columnar protocols remain essential.

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.

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 OTAP Data Organization. OTAP transforms OTLP's nested row-oriented structure into a flat columnar layout. For example, metric data becomes:

```
Metric Table (columnar)
|- resource_id: [0,0,1,1,2,2,...]
|- metric_name: [cpu,mem,cpu,mem,...]
|- timestamp: [t1,t2,t3,t4,...]
|- value: [0.5,8.2,0.6,7.9,...]
+- attributes: [dict_ref,dict_ref,...]
```

Each column stores only values for a single field, enabling specialized compression and efficient access patterns.

3.2.3 Dictionary Encoding in OTAP. Dictionary encoding is crucial to OTAP's compression efficiency. Instead of repeating string values (like metric names or attribute keys), OTAP:

(1) Assigns each unique string an integer ID
(2) Transmits the dictionary (string → ID mapping)
(3) Replaces string values with compact integer references

OTAP supports three dictionary modes:

- **Stateful (Delta Dictionary):** Dictionary is maintained across batches. Only new entries (deltas) are transmitted. Most efficient for stable schemas.
- **Dictionary per Batch:** Each batch includes its own complete dictionary. Provides self-contained batches at the cost of redundant dictionary transmission.
- **No Dictionary:** Strings are transmitted directly. Essentially disables one of OTAP's key compression mechanisms.

Our experiments show that dictionary encoding is fundamental to OTAP's compression advantage, i.e., without it, OTAP can actually perform worse than OTLP with standard compression.

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 but are excluded from evaluation to prevent overfitting.
(3) **Training:** OpenZL trains the compressor using our samples specifying encoding strategies for each field.

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 Time:** Time required to compress a payload. Critical for high-throughput telemetry systems where compression occurs on the hot path.
- **Decompression Time:** Time required to decompress. 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 four 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 times:** 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.

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 the three dictionary encoding strategies, and also implement dictionary per batch for OTLP.

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:** Our evaluation focuses on the `LineItem` table, the schema's largest relation, which represents the granular details of individual orders and constitutes the bulk of the benchmark's data volume.

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 and test sets. We then generated payloads of varying batch sizes and serialized them to protobuf format.

5.3.2 OTAP Transformation. OTLP payloads are transformed to OTAP using the STEF (Structured Telemetry Export Format) transformer. This process parses OTLP protobuf messages, extracts all data points into flat tables, applies dictionary encoding to categorical fields, and serializes the result to Arrow IPC streaming format. This transformation preserves all data while reorganizing it from a row-oriented to a columnar layout.

5.4 OpenZL Training

For each workload, we implemented OTLP parser variants for metrics and traces, and trained OpenZL compressors on the training samples. Training was performed once per workload type (metrics vs. traces) and reused across batch sizes.

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.

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)	37.17M	1.00	556.3	464.9K	79.94	7.0	375.7K	98.94	5.6
OTAP (Arrow)	5.71M	6.50	85.5	220.9K	168.29	3.3	221.3K	167.91	3.3
Hipstershop Traces									
OTLP (Proto)	88.98M	1.00	1027.6	2.89M	30.80	33.4	2.74M	32.47	31.7
OTAP (Arrow)	8.83M	10.08	102.0	1.78M	50.01	20.5	1.77M	50.15	20.5
Astronomy Metrics									
OTLP (Proto)	299.65M	1.00	380.9	6.26M	47.87	8.0	4.49M	66.78	5.7
OTAP (Arrow)	64.79M	4.62	82.4	3.31M	90.66	4.2	3.26M	91.92	4.1
Astronomy Traces									
OTLP (Proto)	62.84M	1.00	979.6	2.95M	21.27	46.1	2.54M	24.70	39.7
OTAP (Arrow)	13.63M	4.61	212.4	2.23M	28.19	34.7	2.23M	28.12	34.8
TPC-H LineItem									
Proto	84.56M	1.00	140.8	22.72M	3.72	37.8	17.86M	4.73	29.7
Arrow	82.01M	1.03	136.5	18.93M	4.47	31.5	–	–	–

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

Hypothesis H1 (OpenZL improves row-oriented compression): *Supported.* OpenZL mostly achieves higher compression ratios for OTLP compared to standard zstd across all workloads, especially at larger batch sizes.

Hypothesis H2 (Columnar format aids compression): *Strongly supported.* OTAP with delta dictionary encoding and zstd consistently outperforms OTLP+zstd, as well as OTLP+OpenZL validating the OpenTelemetry community's motivation for developing OTAP. The columnar organization provides 1-2x improvement in compression ratio over row-oriented format with the same compressor.

Hypothesis H3 (Format-aware compression bridges the gap): *Not well supported.* Even batch sizes above 10000 data points, OTLP+OpenZL fails to achieve compression ratios comparable to OTAP+zstd (delta dictionary).

6.2 Compression and Decompression Speed

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.

6.3 Dictionary Encoding Impact on OTAP

To understand the fundamental mechanisms behind OTAP's compression efficiency, we evaluated three dictionary encoding strategies:

(1) **Stateful (Delta Dictionary):** Maintains dictionary across batches, transmitting only new entries
(2) **Dictionary per Batch:** Each batch includes a complete dictionary
(3) **No Dictionary:** Disables dictionary encoding entirely

Critical finding: Dictionary encoding is fundamental to OTAP's compression advantage. Without dictionary encoding, OTAP+zstd actually performs *worse* than OTLP+zstd, demonstrating that the columnar format alone is insufficient—the combination of columnar organization and dictionary encoding is what enables superior compression. Therefore the success of columnar protocols depends not just on data reorganization, but on complementary encoding strategies that exploit the resulting data locality. Following this, we sought to see if using dictionary encoding within OTLP will be beneficial, and we did see some gains in compression speed (but not in compression ratio) by using dictionary per batch. This motivates investing some effort in making delta

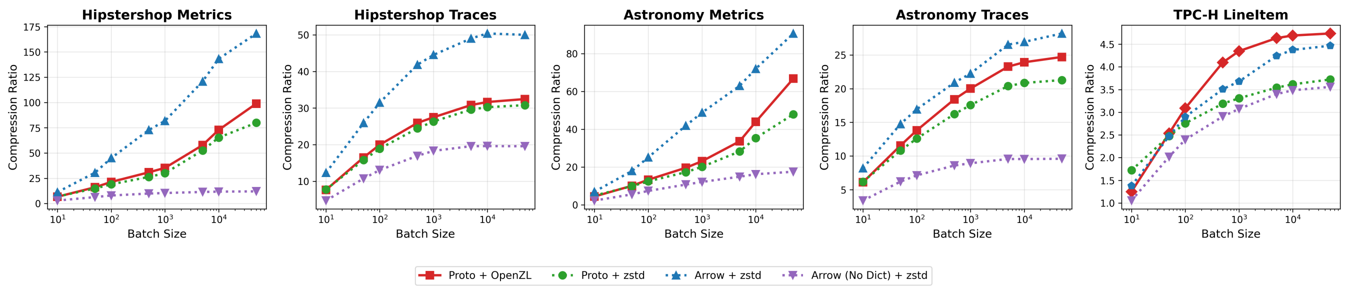

Figure 1: Compression ratio vs. batch size across all workloads.

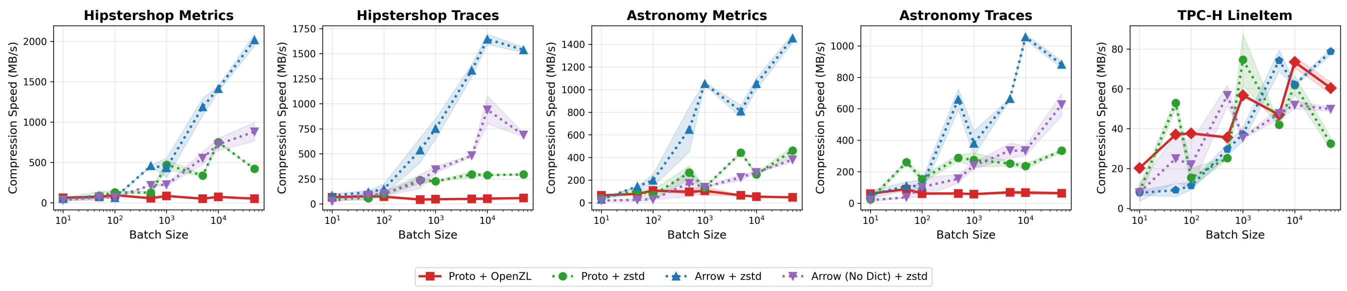

Figure 2: Compression speed vs. batch size across all workloads.

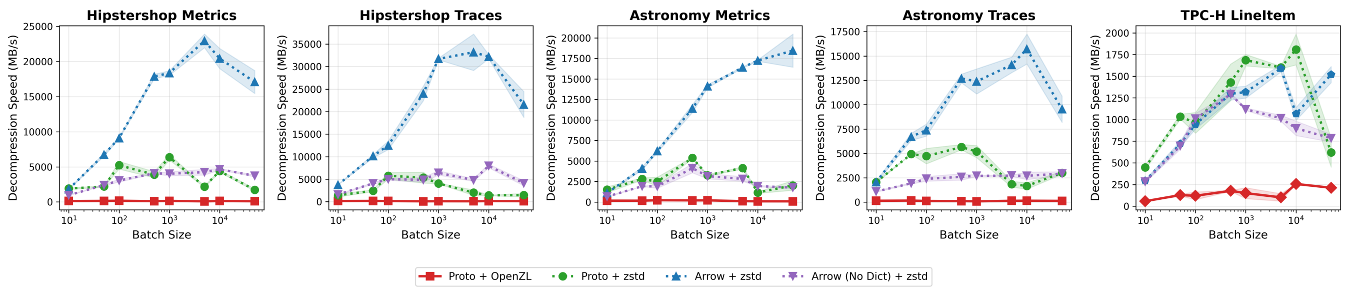

Figure 3: Decompression speed vs. batch size across all workloads.

dictionary work for row formats. This also seems to be an open problem that the OTel community is looking at.

6.4 Workload-Specific Analysis

While metrics and traces showed no consistent performance differences, TPC-H consistently achieved strong performance with OpenZL. This likely results from two complementary factors: OTAP heavily exploits columnar deduplication through dictionary encoding, and the OTel community has extensively optimized OTAP for telemetry workloads, explaining its superior performance on OTel data compared to generic analytical datasets like TPC-H.

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. Apache Parquet [1] and ORC [7] demonstrated that columnar formats dramatically improve compression ratios and query performance for analytical workloads. Apache Arrow [5] extended these ideas to in-memory processing and inter-process communication.

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:

Time-series databases: Gorilla [8] introduced delta-of-delta encoding and XOR-based compression for time-series metrics. These techniques exploit temporal locality and value similarity in metrics data.

Log compression: LogZip [4] 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:

XML compression: XMill [3] and XPRESS [6] separate structure from content in XML documents, enabling better compression of both components.

Protocol Buffer optimization: Cap'n Proto's [9] "packing" compression removes sequences of zero bytes (from unset fields, padding, and unused space) 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, bridging the gap between general-purpose and domain-specific compression. However, we found that OTAP's exemplar performance remains difficult to surpass, suggesting that substantial engineering effort is required to achieve optimal results. Despite this limitation, OpenZL serves as a viable alternative for non-OTel workloads. The principal contribution of this work lies in demonstrating that format-aware compression can approach the compression benefits of columnar formats without requiring data reorganization, though it cannot replicate the query-time and processing advantages inherent to columnar storage architectures.

7.4 Practical Deployment Considerations

Organizations evaluating these approaches should consider:

(1) **Migration cost vs. benefit:** Protocol migration involves not just code changes but operational risk, testing, and potential service disruption. Format-aware compression can be deployed as a drop-in replacement for existing compressors with minimal infrastructure changes.
(2) **Training overhead:** OpenZL requires representative training data and periodic retraining as workload characteristics evolve. Organizations must establish processes for training data collection and compressor updates.

Ultimately, the choice between format-aware compression and protocol migration depends on an organization's specific constraints and priorities. Our work provides empirical evidence to inform this decision, demonstrating that sophisticated compression techniques can bridge much, though not all of the efficiency gap between row-oriented and columnar protocols.

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] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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