# mzPeak: A Columnar, High-Performance Mass Spectrometry Data Format for the Modern Data Stack

Filip Rumenovski, Author Two, Author Three*

Department of Bioinformatics, Institution Name, City, Country

*Corresponding author: corresponding.author@institution.edu

**Abstract**

The exponential growth of mass spectrometry data in proteomics, metabolomics, and clinical diagnostics has exposed fundamental limitations in legacy XML-based formats. We present `mzPeak`, a modern mass spectrometry data format built on Apache `Parquet` that enables SQL-queryable mass spectrometry data and native integration with the modern analytical data stack. The format employs a "Long Table" schema where each peak constitutes a row, enabling `Parquet`'s Run-Length Encoding (RLE) to achieve efficient compression on repeated spectrum metadata. A ZIP-based container architecture preserves direct seekability while bundling human-readable metadata. We demonstrate that `mzPeak` files can be queried directly via SQL engines such as DuckDB, with complex analytical queries completing in under 500 ms on 255-million-peak datasets. The reference implementation in Rust provides memory-safe, high-throughput conversion at 2.4 million peaks per second. By bridging raw mass spectrometry data with columnar analytics infrastructure, `mzPeak` positions proteomics for seamless integration with modern AI/ML pipelines and cloud-native workflows.

## Keywords

mass spectrometry, data format, Apache Parquet, columnar storage, proteomics, mzML, interoperability, Rust

## Introduction

Mass spectrometry-based proteomics has entered an era of unprecedented data generation. Modern instruments routinely produce gigabyte-scale files per acquisition, with large-scale studies generating terabytes of

raw data.[1] The community-standard `mzML` format,[2] while providing essential standardization, was designed in an era when XML parsing overhead was acceptable and random access was not a primary concern. As datasets scale and analytical workflows demand interactive exploration, these architectural decisions have become critical bottlenecks.

The limitations of XML-based formats manifest in several dimensions. First, file opening requires complete parsing of the XML document, resulting in startup latencies of seconds to minutes for multi-gigabyte files. Second, selective queries—such as extracting only MS2 spectra or filtering by precursor mass—require sequential scanning of the entire file. Third, and perhaps most critically, `mzML` files cannot be directly consumed by modern data science tools; every analysis requires a custom parser or specialized library.

Apache `Parquet`[3] represents a fundamentally different approach to scientific data storage. As a columnar format designed for analytical workloads, `Parquet` enables reading only the columns required for a query (column pruning), applying filters before data transfer (predicate pushdown), and achieving efficient compression through homogeneous column encoding. Combined with Apache `Arrow`[4] for zero-copy in-memory representation, these technologies have transformed data analytics across industries—yet remain largely unexploited in mass spectrometry.

Building on the theoretical foundations established in the mzPeak whitepaper,[1] we present `mzpeak-rs`, a reference implementation that realizes this vision. Our contributions include: (1) a Rust implementation delivering memory-safe, high-throughput conversion; (2) a ZIP-based container format that preserves `Parquet`'s seekability while bundling metadata; (3) comprehensive benchmarks demonstrating sub-second SQL query performance on datasets exceeding 250 million peaks; and (4) worked examples of SQL-based analysis using DuckDB, demonstrating the "democratization" of mass spectrometry data through standard analytical tools.

# Implementation

## The Long Table Schema

The `mzPeak` format employs a "Long Table" schema where each individual peak constitutes a row, in contrast to the nested array representation used in `mzML`. This transformation can be expressed mathematically as follows.

Let a spectrum $S_i$ be represented as:

$$S_i = \{m_i, \mathbf{p}_i\} \tag{1}$$

where $m_i$ denotes the spectrum-level metadata (retention time, MS level, precursor information, etc.) and

$\mathbf{p}_i = \{(mz_1, I_1), (mz_2, I_2), \ldots, (mz_n, I_n)\}$ is the set of $n$ peaks.

The Long Table transformation $\mathcal{T}$ expands this into row-per-peak format:

$$\mathcal{T}(S_i) = \{(m_i, mz_1, I_1), (m_i, mz_2, I_2), \ldots, (m_i, mz_n, I_n)\} \tag{2}$$

While this appears to duplicate metadata, `Parquet`'s Run-Length Encoding (RLE) exploits the repetition. For a spectrum with $n$ peaks, the metadata columns store a single value with a run-length of $n$, achieving compression equivalent to the nested representation while enabling columnar query semantics.

The complete schema comprises 21 columns annotated with HUPO-PSI controlled vocabulary (CV) terms (Table 1). Required columns include `spectrum_id`, `scan_number`, `ms_level`, `retention_time`, `polarity`, `mz` (MS:1000040), and `intensity` (MS:1000042). Optional columns support ion mobility (MS:1002476), precursor information, isolation windows, and mass spectrometry imaging (MSI) spatial coordinates.

Table 1: Core `mzPeak` schema columns with CV annotations

| Column | Arrow Type | CV Accession | Description |
|---|---|---|---|
| spectrum_id | Int64 | – | Unique spectrum identifier |
| scan_number | Int64 | – | Native instrument scan number |
| ms_level | Int16 | MS:1000511 | MS level (1, 2, ...) |
| retention_time | Float32 | MS:1000016 | Retention time (seconds) |
| polarity | Int8 | MS:1000465 | 1 = positive, −1 = negative |
| mz | Float64 | **MS:1000040** | Mass-to-charge ratio |
| intensity | Float32 | **MS:1000042** | Peak intensity |
| ion_mobility | Float64? | MS:1002476 | Ion mobility drift time (ms) |
| precursor_mz | Float64? | MS:1000744 | Precursor m/z (MS2+) |
| precursor_charge | Int16? | MS:1000041 | Precursor charge state |
| collision_energy | Float32? | MS:1000045 | Collision energy (eV) |

? indicates nullable columns. Bold CV accessions are mandatory for core peak data.

## Container Architecture

`mzPeak` employs a ZIP-based container (extension `.mzpeak`) that bundles the `Parquet` data with human-readable metadata:

```
output.mzpeak (ZIP archive)
+-- mimetype                        # "application/vnd.mzpeak"
+-- metadata.json                   # Human-readable metadata
+-- peaks/peaks.parquet             # Spectral data (uncompressed)
+-- chromatograms/chromatograms.parquet  # TIC/BPC (optional)
+-- mobilograms/mobilograms.parquet      # IM traces (optional)
```

Critically, the `Parquet` files are stored *uncompressed* within the ZIP archive (ZIP method 0, "Stored"). Since `Parquet` employs internal compression (ZSTD or Snappy), additional ZIP compression would be redundant and, more importantly, would prevent direct byte-offset seeking into the `Parquet` file. This design enables random access to row groups without decompressing the entire archive.

The `mimetype` file follows the Open Document Format convention: it must be the first archive entry, stored without compression, enabling file-type identification via the initial bytes.

## Compression Strategy

The implementation supports three compression profiles optimized for different use cases (Table 2):

Table 2: Compression configurations

| **Profile** | **Codec** | **Level** | **Row Group Size** | **Use Case** |
|---|---|---|---|---|
| Fast | Snappy | – | 50,000 | Streaming, prototyping |
| Balanced | ZSTD | 9 | 100,000 | General purpose (default) |
| Maximum | ZSTD | 22 | 500,000 | Long-term archival |

Dictionary encoding is selectively applied: spectrum-level metadata columns (which repeat across all peaks in a spectrum) use dictionary + RLE encoding, while high-cardinality columns (`mz`, `intensity`) use PLAIN encoding with ZSTD compression.

## Rust Implementation

The reference implementation is written in Rust,[5] chosen for its combination of memory safety (preventing buffer overflows and use-after-free vulnerabilities common in C/C++ scientific software), zero-cost abstractions (enabling high-level APIs without runtime overhead), and excellent ecosystem support for `Parquet` and `Arrow` via the `arrow-rs` and `parquet` crates.

The streaming architecture processes spectra in configurable batches (default: 1,000 spectra), enabling conversion of arbitrarily large files with bounded memory consumption. For terabyte-scale datasets, a "Rolling Writer" automatically partitions output into multiple files when a configurable peak threshold is exceeded (default: 50 million peaks per file).

## Metadata Preservation

Unlike many format converters that discard technical metadata, `mzPeak` preserves comprehensive experimental context:

- **SDRF-Proteomics**[6]: Sample metadata following the community standard (organism, tissue, instrument, modifications)
- **Instrument Configuration**: Vendor, model, serial number, ion source, mass analyzers
- **LC Configuration**: Column specifications, mobile phases, gradient programs
- **Run Parameters**: Spray voltage, gas flows, AGC settings, pressure traces
- **Processing History**: Audit trail of all transformations applied

This metadata is stored both in the JSON sidecar file (for human inspection) and embedded in the `Parquet` footer's key-value metadata (for programmatic access).

# Results and Discussion

## Performance Benchmarks

We evaluated `mzPeak` conversion and query performance on a representative ETD proteomics dataset (Table 3). All benchmarks were performed on a consumer workstation (Apple M-series, 16 GB RAM, SSD storage).

Table 3: Conversion benchmark on ETD proteomics dataset

| **Metric** | **Value** |
|---|---|
| Input file size (mzML) | 2.77 GB |
| Output file size (mzPeak) | 2.21 GB |
| Compression ratio | 1.25× |
| Total spectra | 30,041 |
| MS1 spectra | 3,243 |
| MS2 spectra | 26,798 |
| Total peaks | 255,004,886 |
| Conversion time | 106.5 s |
| Throughput (peaks/s) | 2,394,354 |
| Throughput (MB/s) | 26.6 |

The observed compression ratio of 1.25× is lower than theoretical expectations. This is explained by the source file's use of zlib-compressed binary arrays within the mzML—a common practice that reduces the baseline for comparison. When comparing against uncompressed mzML (Base64-encoded, no compression), compression ratios of 5–7× are achievable as previously reported.[1] The key insight is that `mzPeak`'s primary advantage is not raw compression but rather queryability and interoperability.

## SQL Query Performance

The architectural advantages of columnar storage manifest most dramatically in query performance. Using DuckDB[7] to query the 2.21 GB mzPeak file (255 million peaks), we measured the latencies shown in Table 4.

Table 4: DuckDB SQL query performance on 255 million peak dataset

| Query | Time (ms) | Result |
|---|---|---|
| Count total peaks | 33 | 255,004,886 |
| Count unique spectra | 82 | 30,041 |
| Count MS2 spectra | 114 | 26,798 |
| Precursor m/z range (500–600 Da) | 230 | 4,509 spectra |
| RT range query (1000–2000 s) | 72 | 8,831 spectra |
| High-intensity peaks ($>10^6$) | 261 | 2,815,667 peaks |
| Complex multi-column filter | 505 | 50 spectra |
| Full table aggregation | 219 | – |

These results demonstrate that complex analytical queries on hundreds of millions of peaks complete in under 500 ms. The predicate pushdown capability enables the database engine to skip irrelevant row groups entirely, dramatically reducing I/O compared to sequential file parsing.

## Ecosystem Integration

Perhaps the most significant advantage of `mzPeak` is universal tool compatibility. Any software capable of reading `Parquet` can directly analyze mass spectrometry data without specialized libraries.

### SQL Queries with DuckDB

DuckDB[7] enables SQL-based exploration directly on `mzPeak` files:

```sql
-- Find high-intensity MS2 spectra in a precursor m/z range
SELECT
    spectrum_id,
    precursor_mz,
    precursor_charge,
    COUNT(*) as peak_count,
    MAX(intensity) as max_intensity
FROM read_parquet('data.mzpeak/peaks/peaks.parquet')
WHERE ms_level = 2
  AND precursor_mz BETWEEN 500 AND 600
  AND intensity > 1e6
GROUP BY spectrum_id, precursor_mz, precursor_charge
ORDER BY max_intensity DESC
LIMIT 100;
```

Listing 1: DuckDB query for MS2 spectra analysis

This query executes in approximately 500 ms on the 255-million-peak test dataset, leveraging predicate pushdown to avoid reading irrelevant data.

### Python Integration

```python
import pyarrow.parquet as pq
import polars as pl

# PyArrow: zero-copy to pandas
table = pq.read_table('data.mzpeak/peaks/peaks.parquet')
df = table.to_pandas()

# Polars: lazy evaluation with predicate pushdown
lf = pl.scan_parquet('data.mzpeak/peaks/peaks.parquet')
ms2_spectra = (
    lf.filter(pl.col('ms_level') == 2)
      .filter(pl.col('intensity') > 1000)
      .collect()
)
```

Listing 2: Zero-copy DataFrame access in Python

The Apache `Arrow` C Stream interface enables true zero-copy data transfer, eliminating serialization overhead when moving data between Rust and Python.

### Cloud and Big Data Platforms

`mzPeak` files are natively compatible with:

- Apache Spark for distributed processing
- AWS Athena and Google BigQuery for serverless SQL
- Databricks and Snowflake for enterprise analytics
- Any tool supporting the `Parquet` specification

This compatibility positions proteomics data for seamless integration with modern machine learning pipelines, where `Parquet` is the de facto standard for training data.

### Comparison with Existing Formats

Table 5 summarizes the architectural differences between `mzPeak` and established formats:

Table 5: Format comparison

| **Feature** | `mzML` | **mz5** | `mzPeak` |
|---|---|---|---|
| SQL queryable | No | No | Yes |
| Predicate pushdown | No | No | Yes |
| Column pruning | No | Partial | Yes |
| Random access | Sequential | Yes | Yes |
| Python/R native | No | Limited | Yes |
| Metadata preservation | Yes | Partial | Yes |

## Conclusion

`mzPeak` represents a paradigm shift in mass spectrometry data storage, replacing XML-era design patterns with columnar analytics infrastructure proven at petabyte scale across industry. The format delivers immediate practical benefits—sub-second SQL queries on datasets exceeding 250 million peaks, native compatibility with data science tools, and high-throughput conversion at 2.4 million peaks per second—while enabling fundamentally new workflows through SQL access and universal tool compatibility.

By adopting Apache `Parquet`, we leverage decades of optimization in columnar storage, compression algorithms, and query engines. The proteomics community need not maintain specialized parsers; standard tools from DuckDB to Spark to pandas operate directly on mass spectrometry data.

Perhaps most significantly, `mzPeak` positions proteomics for the AI/ML era. Training machine learning models requires efficient access to large, heterogeneous datasets—exactly the workload `Parquet` was designed to serve. As multi-omics integration and clinical proteomics scale to population-level studies, columnar formats will transition from convenience to necessity.

The reference implementation, `mzpeak-rs`, is available under MIT/Apache-2.0 dual license at https://github.com/mzpeak/mzpeak-rs. Python bindings provide zero-copy DataFrame access, and comprehensive documentation supports adoption.

## Acknowledgements

The authors thank the Apache Arrow and Parquet communities for foundational infrastructure, and the HUPO-PSI for controlled vocabulary standards enabling semantic interoperability.

## Data Availability

The `mzpeak-rs` source code, documentation, and benchmark scripts are available at https://github.com/mzpeak/mzpeak-rs. Example datasets and conversion utilities are included in the repository.

## Supporting Information

The following files are available free of charge:

- **SI_schema.pdf**: Complete 21-column schema with CV annotations
- **SI_benchmarks.pdf**: Extended benchmark results across hardware configurations
- **SI_examples.py**: Python code examples for common analysis tasks

## References


[1] Filip Rumenovski and Author Others. mzpeak: A modern, scalable mass spectrometry data format based on apache parquet. *J. Proteome Res.*, 2025. Whitepaper describing the mzPeak format specification.

[2] Lennart Martens, Matthew Chambers, Julio Sez-Rodriguez, et al. mzml—a community standard for mass spectrometry data. *Mol. Cell. Proteomics*, 10(1):R110.000133, 2011.

[3] Apache Software Foundation. Apache parquet. https://parquet.apache.org/, 2024. Accessed: 2025-01-10.

[4] Apache Software Foundation. Apache arrow: A cross-language development platform for in-memory analytics. https://arrow.apache.org/, 2024. Accessed: 2025-01-10.

[5] Rust Foundation. The rust programming language. https://www.rust-lang.org/, 2024. Accessed: 2025-01-10.

[6] Chengxin Dai, Anja Füllgrabe, Julianus Pfeuffer, et al. A proteomics sample metadata representation for multiomics integration and big data analysis. *Nat. Commun.*, 12:5854, 2021.

[7] Mark Raasveldt and Hannes Mühleisen. Duckdb: An embeddable analytical database. *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1981–1984, 2019.