Bullion: A Column Store for Machine Learning

Gang Liao
Ye Liu
Jianjun Chen
Daniel J. Abadi
CIDR 2025, Amsterdam, Netherland

Outlines

- Background & Motivation
- Challenges Posed by Modern ML Workloads
  - Data Compliance
  - Native Handling of Vectors
  - Wide Table Projection
  - Storage Quantization
  - Multimodal Data Storage
  - Cascading Encoding Framework
- Conclusion

Transition to Columnar Storage

- **Row-Oriented Storage (Pre-2000s):**
  - Data stored row-by-row.
  - **Efficient for** transactional workloads (e.g., CRUD operations).
  - **Poor for** analytics due to high I/O costs and limited parallelism.

- **Columnar Storage (2000s Onwards):**
  - Projects like C-Store and X100 shifted to column-by-column storage.
  - **Optimized for analytics** with better compression and parallelism.

Strengths of Column Stores

- **Efficient Compression**: Better ratios, direct operations on compressed data.
- **Faster Queries**: Skip irrelevant columns, leverage SIMD for parallelism.
- **Broad Adoption**: Formats like **Parquet** and **ORC** are now industry standards, widely supported by query engines in Lakehouse.

Machine Learning Workloads: A Rising Demand

- **Key Use Cases**
  - **Ads & Recommendations**: Large-scale feature sets for personalized ranking.
  - **Generative AI**: Multimodal data (text, images, video) for training and serving.
  - **LLM-powered Apps**: High-dimensional embeddings, real-time vector search.

- **Limitations of Existing Columnar Formats**
  - **Data Compliance**: High cost for in-place deletes.
  - **Wide, Sparse Tables**: Inefficient metadata handling.
  - **Vector Support**: Poor optimization for vector and embeddings.
  - **Storage Efficiency**: Limited quantization options for ML features.
  - **Multimodal Data**: Fragmented storage and access patterns.
  - **Encoding Framework**: Rigid, non-modular encoding schemes.

- **Objective**: Bullion is designed to address these challenges.

Challenge 1 - Data Compliance (1/3)

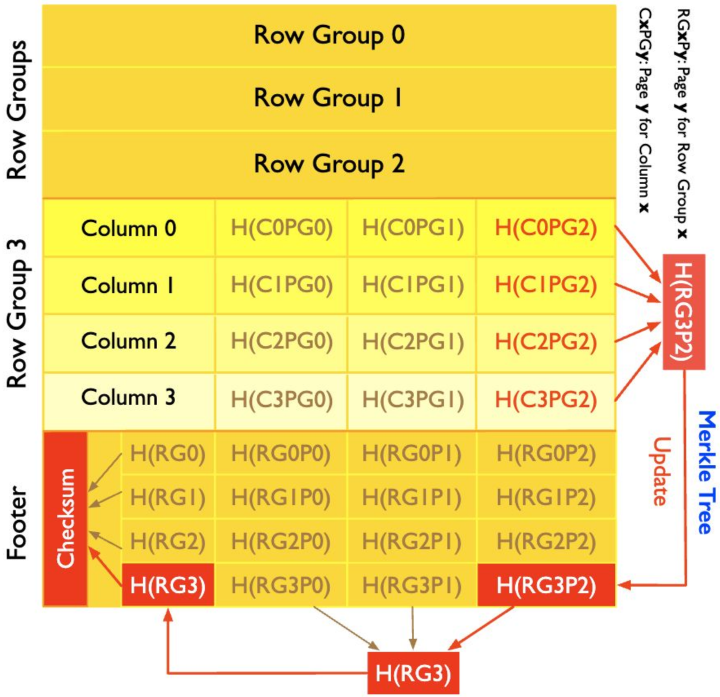
- **Problem**
  - Privacy regulations (e.g., GDPR, CCPA) require **timely and physical deletion**.
  - Traditional columnar storage **struggles** with efficient and compliant deletions.
  - Deleting just **5%** of non-compliant data requires rewriting **hundreds of petabytes** per month at TikTok.
- **Why**
  - **Fragmentation**: Each column in a row is stored separately, requiring multiple modifications for a single row deletion.
  - **Block-based compression** complicates direct modifications of individual rows.
- **Existing Approaches**
  - **Traditional approach**: Full file rewrites consuming **~20x more I/O** than necessary.
  - **Out-of-Place Deletes**: Marks data as "hidden" but does not physically delete it..
  - **Impact**: ByteDance's CN region tables **exceed 1EB**, making rewriting prohibitively expensive.

Challenge 1 - Data Compliance (2/3)

- **Bullion:** Ensures compliance while minimizing file rewrites.
  - **In-Place Deletes**
    - Selective row-level physical deletion
    - **Encoding-aware masking** operations
    - No full decompression needed
  - **Example**
    - Bit-Packed Encoding:
      - Direct bit masking of fixed-width values
    - Dictionary Encoding:
      - Adds special mask value to dictionary
      - Updates reference to mask value
    - RLE Encoding:
      - Selective value masking & Updates run counts
    - FOR-delta Encoding:
      - Preserves base values & Masks deltas directly

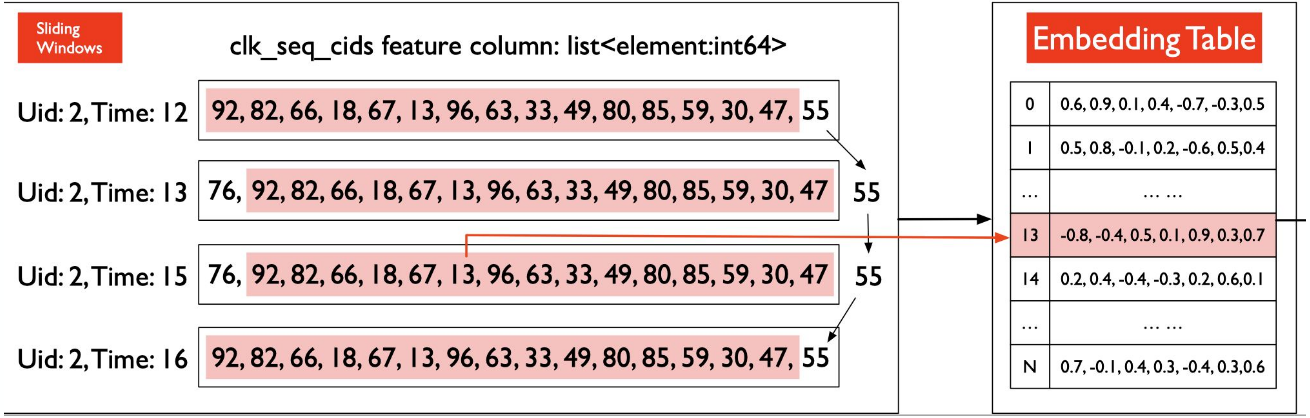
Challenge 1 - Data Compliance (3/3)

- **Bullion:** Ensures compliance while minimizing file rewrites.
  - **Integrity via Merkle Tree**
    - **Page-Level Checksums**: H(C0PG0), H(C1PG0), etc.
    - **Row Group Checksums**: H(RG0), H(RG1), etc.
    - **File-Level Checksum**: Computed from row group checksums
  - **Benefits**
    - **Minimal I/O**: Only read affected pages
    - **Fast Verification**: Hierarchical structure



Challenge 2 - Native Handling of Vectors

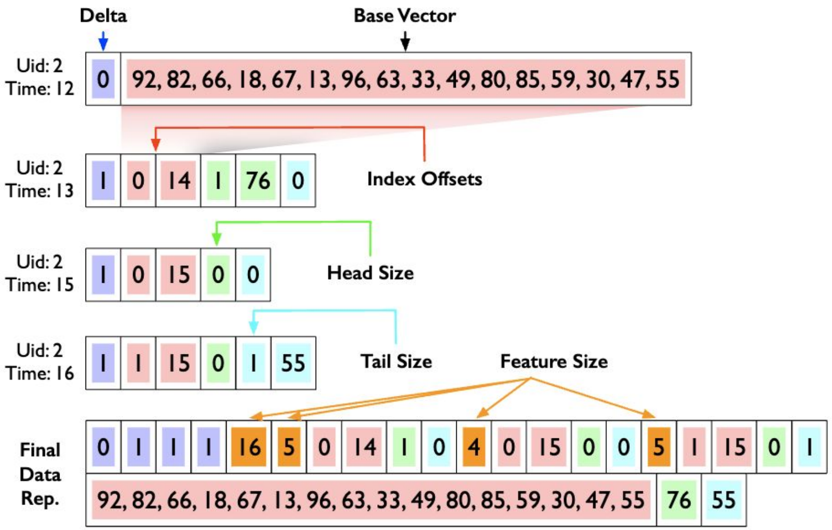
- **Background**
  - Personalization ML workloads often involve **vector-based sparse features**.
  - These vectors exhibit **sliding window patterns**, where successive values change minimally.



This feature is used for tracking user interactions with advertising campaigns over time

Challenge 2 - Native Handling of Vectors

- **Problem**
  - Existing columnar formats (e.g., Parquet, ORC) support **delta encoding only for primitive types** (e.g., INT, BIGINT).
  - **Inefficient** storage and high I/O costs for vectors with repetitive elements.
- **Bullion** optimized native support for vector types moving forward.



Challenge 3 - Wide Table Projection

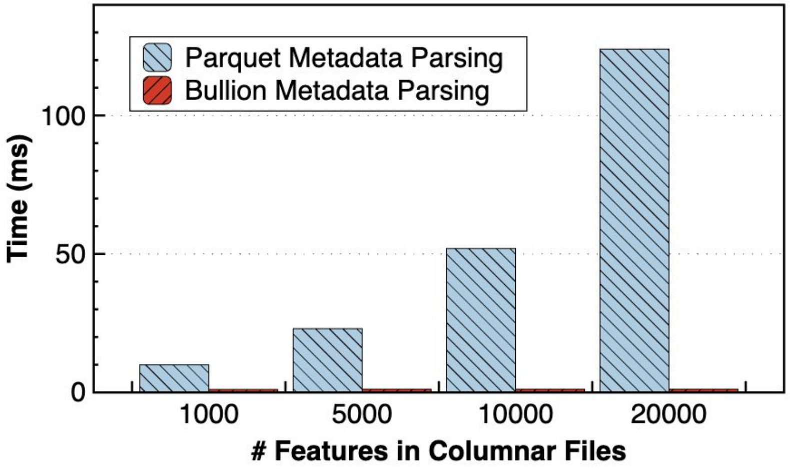
- **Problem**
  - Columnar formats **were designed for SQL workloads** (e.g., sorting, grouping, aggregations).
  - Current formats require **full deserialization of metadata** before column access.
  - Modern ML workloads:
    - Feature counts often **exceed 10,000**, with **most features rarely accessed** in ByteDance ads tables.
    - **High Metadata Overhead**: Metadata access time scales linearly with the number of columns, increasing query latency.

Column Type	**# Columns**
list<int64>	**16,256**
list<float>	812
list<list<int64>>	277
struct<list<int64>, list<float>>	143
struct<list<int64>>	120
struct<list<binary>>	46
struct<list<float>>	29
struct<list<binary>, list<binary>>	18
struct<list<double>>	10
list<binary>	8
struct<list<list<int64>>>	5
struct<list<binary>, list<float>>	5
string	3
int64	1

Table 1. Statistical breakdown of column types in an Ad Parquet file.

Challenge 3 - Wide Table Projection

- **Solution**
  - Directly accesses buffer values from file footers, **eliminating the need for deserialization**.
  - Keeps metadata **parsing time flat**, even for extremely wide tables.
- **Performance Highlights**
  - A consistent parsing time (<2ms) regardless of the number of features.



Challenge 4 - Storage Quantization

- **Problem**
  - High storage and memory costs for dense embeddings and features in Recommender systems and LLMs.
    - **Strict Production Constraints**:
      - Limited storage prevents adding new features and expanding embeddings.
      - High costs for infrastructure and reduced model capabilities.
    - **Sparse Features**: Integer-heavy data contributes significantly to the storage footprint.

Challenge 4 - Storage Quantization

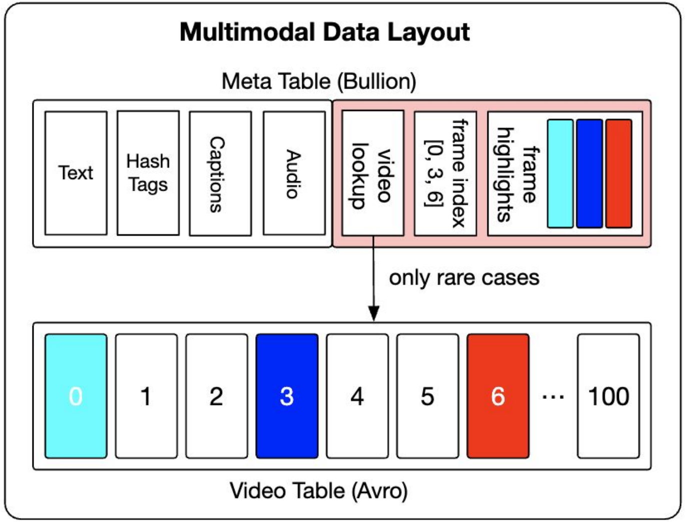
- **Solution**
  - **Feature Quantization**
    - Converts high-precision FP32 embeddings to compact formats
    - Reduces storage, disk I/O, and memory costs while maintaining accuracy.
  - **Mixed-Precision Strategy**
    - Dynamically adjusts precision levels based on feature sensitivity.
  - **Opportunities**
    - **Native support** for reduced-precision formats (e.g., BF16, FP16).
    - **Dual-column strategy** for critical models to maintain FP32 accuracy.

Challenge 5 - Multimodal Data Storage

- **Problem**
  - LLM pre-training requires integration of diverse data types (**text, images, audio, video**)
  - Current dual-table approach architecture:
    - Design Rationale:
      - Meta tables (columnar): Optimized for metadata queries and analytics
      - Media tables (row-oriented): Better for large binary content storage
- **Limitations of Current Approach**
  - **I/O Fragmentation**:
    - Training requires constant switching between tables
    - Each media access needs metadata lookup first
  - **Quality-Based Selection Issues**:
    - Random access patterns when filtering by quality scores
    - Cannot leverage sequential read benefits

Challenge 5 - Multimodal Data Storage

- **Solution**
  - Store critical video frames **directly** in column format at **reduced resolution**.
  - Maintain video indices in meta table for rare full-resolution access.
  - Quality-aware organization: **presort data** by quality scores.
  - **Benefits**:
    - Unified access through columnar storage.
    - Reduced I/O fragmentation.
    - Efficient access to high-quality training samples.



Challenge 6 - Cascading Encoding Framework

- **Problem**
  - ML workloads primarily use integer and floating-point data.
  - Current formats (e.g., Parquet, ORC):
    - Implement **limited subset of encoding** schemes.
    - **Tightly couple** encoding methods.
    - **Lack unified interfaces** for independent use.
  - Growing search space for optimal encoding combinations.
- **Solution**
  - Independent encoding module with cascading capabilities:
    - Built on insights from Nimble and BtrBlocks
    - **Universal design**: compatible with all columnar formats
    - **Pluggable architecture** enables format-agnostic integration
  - Modular, composable interfaces for encoding selection:
    - Mix and match different encoding schemes
    - Easy integration with existing columnar formats
  - Selective use of block compression for rarely accessed columns

Conclusion

Bullion: A Column Storage for Machine Learning

- **Key Contributions**
    - Hybrid deletion compliance mechanism (**50x I/O reduction**)
    - Optimized sparse feature encoding (**20,000+ columns supported**)
    - Fast wide-table projection (**2ms vs 100+ms metadata access**)
    - Storage-level feature quantization (**2-4x space savings**)
    - Quality-aware multimodal data organization
    - Unified cascading encoding framework
- **Real-World Impact**
    - Powering next-gen ML infrastructure:
        i. Ads & Search & Recommendation Systems
        ii. Generative AI & LLMs

THANKS

ByteDance字节跳动