YOUMU: Efficient Columnar Data Pipeline for LLM Training

Tianle Zhong [1] Jiechen Zhao [2] Qiang Su [3] Geoffrey Fox [1]

[1]University of Virginia

[2]University of Toronto

[3]The Chinese University of Hong Kong

May 14, 2025

What Format is Your LLM Dataset?

Large language model (LLM) training datasets are web-scale.

- PiB-level raw data; TiB-level cleaned data.
- Massive data preprocessing.
- Needs to reside on costly high performance storage.

Many training frameworks and datasets are using JSONL, but we know that it is not an ideal choice for managing such amount of data.

Data preparation in Parquet; Why not Training?

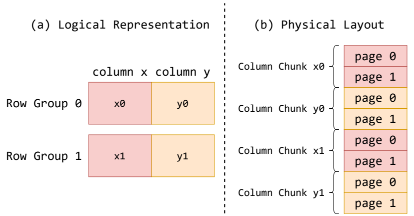


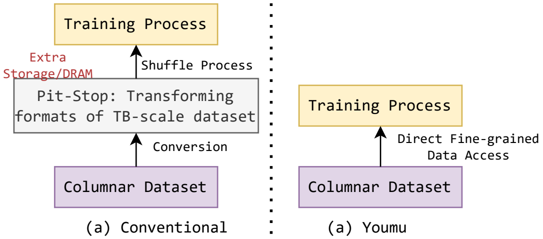
Figure 1: Parquet logical and physical layouts.

With Parquet, data preparation reuses existing data infrastructure for analytical workload.

- Designed for scan-based operations.
- Optimized for block storage.
- High compression rate to reduce storage cost.

However, Parquet is inefficient with random access, a key procedure for data shuffling to ensure model accuracy.

Costly Pit-stops between Preparation and Training



The current practice is to convert & pre-shuffle columnar data into other formats like `JSONL`.

- 3$\times$ more storage capacity needed on costly performant storage.
- Break single truth of data.
- Additional human efforts to keep up with ever evolving datasets with new data and preprocessing refinement.

Existing Solutions

Moving data into distributed memory before training (Ray/Spark).

- High *memory footprint*.
  - Host memory contention with other key functionalities like model checkpointing.
- Complexity with distributed memory
  - Issues with object store over RDMA
  - Network contention with GPU traffic.

Existing Solutions (cont.)

- Disk-backed memory mapping (HuggingFace Datasets).
  - Unsatisfactory *throughput* due to thrashing (page faults).
- Streaming I/O-based local shuffle.
  - Obvious model accuracy loss due to limited *shuffle quality*.

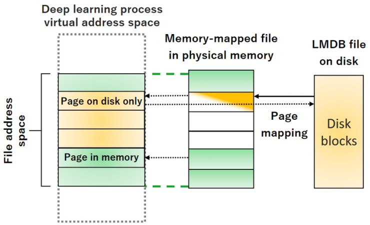


Figure 2: Mmap file I/O [1]

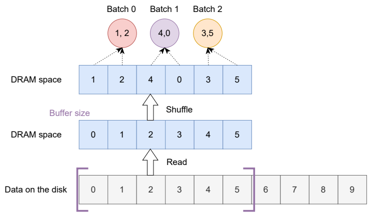


Figure 3: Streaming shuffle.

Our Goals

We aim to stay with one consistent columnar format for training data pipeline, and achieves

- Controlled DRAM footprint (against distributed memory).
  - provide memory resilience for other critical functionalities like tensor offloading and model checkpointing.
- Sufficient throughput (against MMap I/O).
  - Avoid data starvation for GPU utilization.
- High shuffle quality (against local shuffle).
  - Ensure model accuracy.

Observation 1: Marginal Benefits of Caching

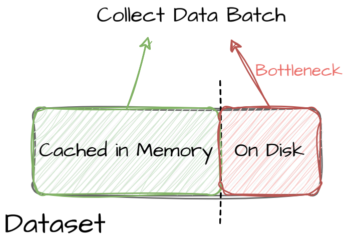


Figure 4: Disk I/O remains as bottleneck

Access to the same row is exactly once per epoch.

- Marginal benefits of caching unless the dataset fully cached.
  - Same observations from [2], [3].

Observation 2: Granularity Gap

Fine-grained shuffling vs. Coarse-grained columnar chunk I/O

- Consume rows but have to read chunks.
  - Tens of KB vs. Hundreds of MB (<0.1% effective bandwidth)
- Waste a lot disk I/O bandwidth.
  - Low goodput. Data fed into CPU but most never used by GPU.

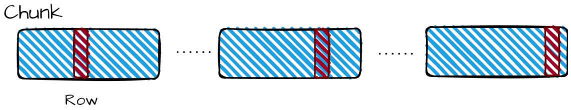


Figure 5: A lot of I/O bandwidth is wasted due to the granularity gap.

YOUMU's Page-level I/O Granularity

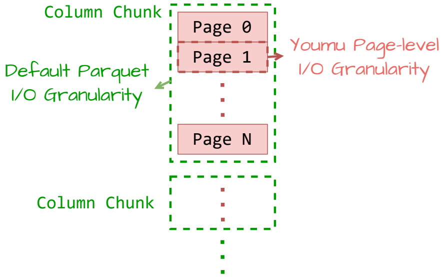

Column Chunk
Page 0
Page 1
Youmu Page-level I/O Granularity
Default Parquet I/O Granularity
Page N
Column Chunk

Figure 6: Page-level I/O.

- Friendly I/O size for SSD.
- Many pages to shuffle.
- Improved I/O goodput.

Challenge: How to navigate dataset to read random page?

Youmu Overview

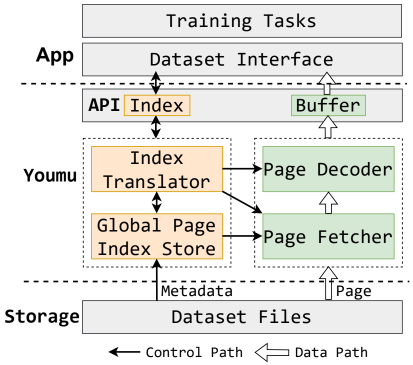


Figure 7: Youmu system overview.

Design highlights:

- No cache, only buffer.
  - From Observation 1.
- Fine-grained page-level I/O.
  - From Observation 2.
- Practical compatibility.
  - Directly works with widely adopted Parquet.

Control Path: Global Page Index

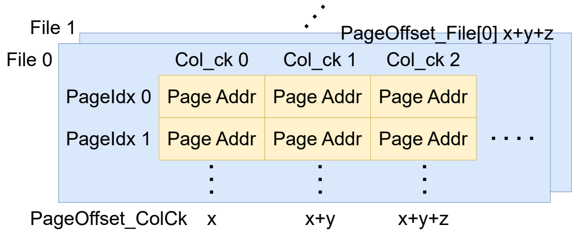


Figure 8: Global page index.

- A 3-dim index across file, column chunk and page locations.
  - Lightweight initialization since only metadata needed.
- Layered binary search-based index translation.
  - Given random page order, page data locations are searched layer-by-layer based on page number offsets of each layer.

Data Path

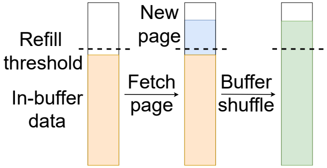


Figure 9: Aggressive buffer shuffle.

- Aggressive buffer shuffle.
  - Based on the fact: page size $<$ buffer size.
  - Shuffle rows in buffer upon the arrival of each new page.
  - Better shuffle quality than standard buffer shuffle at initialization.

Implementation

- Rust runtime.
  - Based on official Rust implementation of Apache Parquet and Arrow.
- Python APIs.
  - Can be directly integrated with PyTorch Dataset interface.
  - User-defined in-memory pipelines for last-mile preprocessing.
  - Zero-copy conversion to various dataframe and tensor formats enabled by Arrow memory model ecosystem.
- Full shuffle support.
  - Support full shuffle by extracting a random row from retrieved page.
  - Sacrifice throughput for perfect shuffle when needed.

Evaluation: Model Accuracy

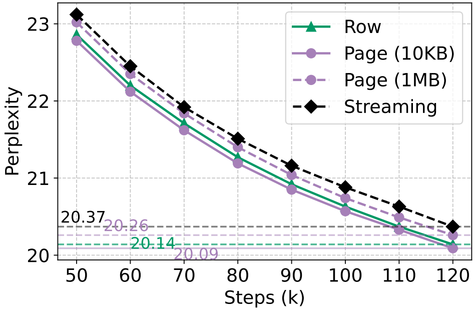


Figure 10: Model accuracy at different training steps

Key Take-away

- A small page (10KB) achieves competitive quality to row shuffling.
- A big page (1MB) still outperforms streaming shuffle.

Evaluation: Memory footprint & Latency

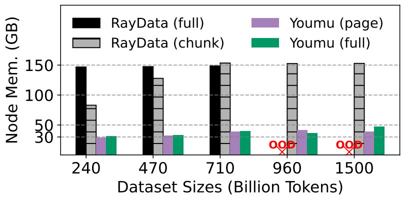


Figure 11: Memory footprint per node against dataset sizes on 16 nodes.

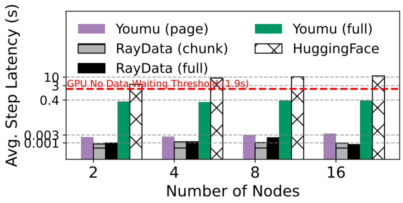


Figure 12: Iteration batch latency against a 200B dataset.

TL;DR

Youmu can achieve sufficiently low latency to avoid GPU idleness while keeping very low memory overheads

Conclusion

- YOUMU preserves I/O and memory efficiency for LLMs training on columnar data storage with high shuffle quality and throughput.
  - Abandon use of cache to reduce memory overheads.
  - Page-level I/O to improve goodput.
  - KB-level page size provides sufficient shuffle quality.

References

[1] A. Haderbache, S. Stanovnik, M. Miwa, and K. Nakashima, “Design of a data supply mechanism for distributed deep learning,”, Jul. 2017.

[2] Y. Zhu, F. Chowdhury, H. Fu, *et al.*, “Entropy-aware i/o pipelining for large-scale deep learning on hpc systems,” in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2018, pp. 145–156. DOI: 10.1109/MASCOTS.2018.00023.

[3] A. V. Kumar and M. Sivathanu, “Quiver: An informed storage cache for deep learning,” in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, Santa Clara, CA: USENIX Association, Feb. 2020, pp. 283–296, ISBN: 978-1-939133-12-0. [Online]. Available: https://www.usenix.org/conference/fast20/presentation/kumar.

Thank You for Your Attention

Q & A Time! :)

Evaluation: Initialization Time (Shuffle Wall)

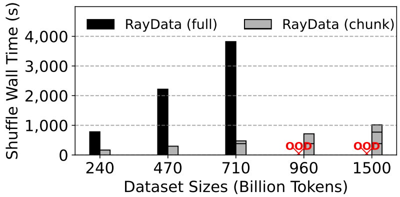


Figure 13: Shuffle wall time required by Ray Data on 16 nodes.

As a comparison, YOUMU only requires less than 10 seconds at initialization to fetch metadata and build global page index.