YOUMU: EFFICIENT COLUMNAR DATA PIPELINE FOR LLM TRAINING

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

ABSTRACT

Large language models (LLMs) training is extremely data-intensive, often involving over trillion-level tokens. Although LLM datasets are usually ingested and stored in columnar formats, they often need to be converted into another format for training, which incurs significant storage and maintenance costs due to extra data copies. While eliminating the conversion would save tens of terabytes of space in costly high performance storage, this work identifies challenges that drive us to re-think the entire data pipeline. Without conversion, we find that fine-grained random access patterns incur hundreds of times efficiency drops. Specifically, the existing data pipelines have two fundamental drawbacks: (1) They cannot efficiently support directly digesting data in columnar format due to default coarse-grained I/O; (2) Solutions to the first drawback sacrifice memory footprint to cache datasets. In this paper, we present YOUMU, a new data pipeline that directly feeds fine-grained columnar data into GPUs, enabling cost-efficient LLM training. Meanwhile, YOUMU maintains high training accuracy, whose perplexity outperforms widely adopted local shuffle by reducing 0.3-0.7 for pretraining. Compared to performance-optimal state-of-the-art, distributed memory-based pipelines, YOUMU achieves comparable throughput with ∼80% less memory footprint.

1 INTRODUCTION

Large language models (LLMs) training relies on huge datasets, often sourced from web-scale repositories (Crawl, 2023; Zhong et al., 2023; Li et al., 2023). Satisfying such demand comes with high storage costs on extensive datasets. Additionally, to generate high-quality training data, intensive processing is required at the data preparation phase (Together Computer, 2023; Penedo et al., 2023; Miao et al., 2024; Tang et al., 2024). To meet the above two requirements, columnar data formats, such as Parquet (Apache, 2023), have become popular in the early stages of data preparation (*e.g.,* storage and cleaning). Benefits brought by columnar formats include high compression ratios and efficient scan-based operations (HuggingFace, 2024c).

As for columnar formats, data is stored and read in large continuous chunks to fully exploit the high performance of storage devices. However, we observe that when directly feeding training datasets under columnar formats into GPUs, retrieving the dataset can become the bottleneck of LLM training, degrading I/O performance by more than 100×. We find that the root cause of such inefficiency is the *granularity mismatch* between (1) fine-grained random access required by GPUs, and (2) coarse-grained columnar chunk-based I/Os supported by current systems. Note that the requirement of a fine-grained random access pattern results from shuffling, which offers mandatory randomness of dataset accesses. Random shuffling is necessary for achieving high model training accuracy (Meng et al., 2019; Gorbunov et al., 2020).

To address this inefficiency problem, prior work adds an extra step between data preparation and training runtime, i.e., converting columnar into other formats that are more random access-friendly. However, such *format transformation* in the preparation phase is costly in the following two aspects. First, the conversion acts as extra "pit-stops" in the data pipeline, resulting in significant redundant storage costs on data copies. For example, converting the 43 TB FineWeb dataset from Parquet to JSON requires approximately 95 TB of additional storage capacity (Penedo et al., 2024). Second, frequent conversions add human effort for maintenance (e.g., version control of differently-formatted datasets) during this pit-stop, leading to higher complexity in end-to-end LLM data management.

In this work, our design goal is to avoid costly format transformation and keep the data format consistent during the entire LLM training data pipeline. Data collected and stored in columnar formats can be directly used by any training

[1]Department of Computer Science, University of Virginia, Charlottesville, VA, USA. [2]Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada. [3]Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong SAR, China.. Correspondence to: Qiang Su <jacksonsq97@gmail.com>, Geoffrey Fox <vxj6mb@virginia.edu>.

Proceedings of the 8th MLSys Conference, Santa Clara, CA, USA, 2025. Copyright 2025 by the author(s).

processes with high efficiency. To realize this goal, the core challenge lies in the inefficiency of random access to columnar data. There are three existing approaches to tackle this challenge, while each of them has fundamental drawbacks. The first approach trades memory footprint for lower storage costs. It prefetches a large portion of the dataset from storage into DRAM and uses DRAM as a cache for efficient random access during shuffling (Zhu et al., 2018; Luan et al., 2023). However, this approach introduces a considerably high memory footprint up to tens of terabytes, which often leads to impracticality as modern LLM dataset sizes keep increasing. Meanwhile, a high memory footprint also constrains the memory for other key tasks in LLM training systems, such as checkpointing (Wang et al., 2023) and tensor offloading (Ren et al., 2021). The second approach relies on memory-mapped I/O to directly access datasets on disk (HuggingFace, 2024a). But it introduces heavy cache thrashing in the OS kernel (Choi et al., 2017; Mohan et al., 2021b), thus leading to poor I/O throughput. The third approach trades shuffle quality for efficiency (Hambardzumyan et al., 2022; HuggingFace, 2024b; PyTorch, 2024; TensorFlow, 2024a), at the risk of significantly degrading model accuracy.

We propose YOUMU, an efficient and practical LLM training data pipeline that (1) enables LLM training to take in columnar data efficiently, (2) requires no format transformation in data preparation phase, (3) preserves high shuffle quality, and (4) improves training efficiency. YOUMU offers the following three insights. First, we observe that the mismatch incurs significant wastes of I/O bandwidth because a large fraction of data in each chunk is fetched but never used by GPUs (i.e., extremely low goodput). Hence, YOUMU proposes finer-grained access on columnar data to improve the goodput. Second, storage systems that hold datasets are often backed by SSDs, highly performant at page level accesses. Thus, YOUMU chooses *pages* as the data retrieval granularity, maximizing SSDs' performance. Third, given a TB-scale LLM datasets, our observation is that accessing the dataset by randomly selecting multi-billion KB-level pages provides much more randomness than randomly accessing several thousand GB-level columnar chunks. Our experiments demonstrate that the proposed page-level random access pattern provides sufficiently high shuffle quality to preserve model accuracy. YOUMU hence further introduces page-level shuffling, which unifies shuffling and disk I/Os both at the page level – eliminating the aforementioned granularity mismatch. As a result, YOUMU avoids format conversions and their resultant extra storage or DRAM cost, i.e., no "pit-stops" on the data pipeline anymore, as shown in Figure 1.

To enable efficient and flexible page data retrieval, YOUMU supercharges columnar data I/Os by a decoupled control plane and data plane. The control plane collects metadata

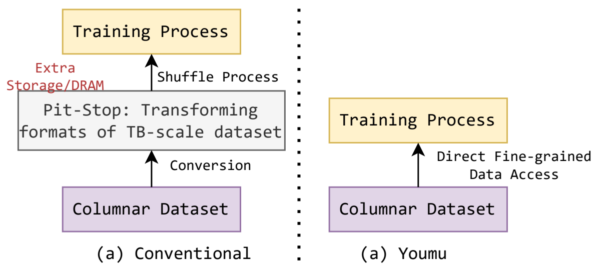

Figure 1. The data path from columnar storage to training process.

of the targeted columns and then builds a *global page index*. This index maps random global page order into the index of file and column chunk, which contains that page and its corresponding physical page address. Hence, the shuffling process can find a random page order's metadata and physical page address on disk by looking up the global page index. The data plane fetches data from the given page address and decodes the compressed data based on corresponding metadata. It also introduces a fixed-size memory buffer to enable (1) additional in-buffer shuffling to further enhance the shuffle quality and (2) overlapping with the training process to hide the data retrieval latency. We expose the decoupled control and data plane with APIs for explicitly managing YOUMU I/Os.

YOUMU is a system that relies on optimized disk I/Os; it primarily targets scenarios where (1) pre-loading the entire dataset into memory is infeasible, or (2) high memory overheads or network contention is unacceptable, degrading training performance.

We implement YOUMU based on open-source Parquet and Arrow libraries in Rust. We conduct extensive experiments to evaluate model accuracy and system overheads, including perplexity measurements, memory footprint, and iteration batch latency. For model accuracy, YOUMU's page-level shuffling achieves comparable accuracy to perfect fully-random shuffling. For system overheads, YOUMU achieves ∼80% less memory footprint compared to the distributed memory-based alternatives, and ensures sufficiently low iteration batch latency for high GPU utilization. To the best of our knowledge, YOUMU is the first practical data pipeline that enables direct fine-grained data access on widely adopted columnar storage for LLM training.

2 COLUMNAR DATA STORAGE FORMAT

Columnar storage formats were developed for efficient analysis of web-scale datasets (Melnik et al., 2010; Armenatzoglou et al., 2022; Google, 2024a). When data is stored in such formats, typical scan-based queries (*e.g.,* selection, projection, and aggregation) can be completed within sec-

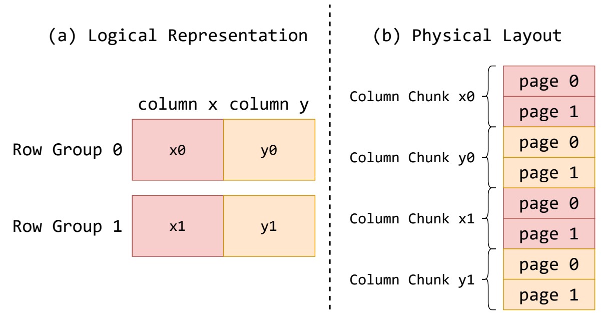

Figure 2. Parquet format's logical representation and physical layout.

onds through workload partitioning and I/O reduction (*e.g.,* data skipping and compression).

In the LLM training data pipeline, the training process only accepts data that are extensively cleaned from raw datasets (*a.k.a.* data preparation), with typical operations like URL filtering, text extraction, quality filtering, deduplication, and anonymizing sensitive information (*e.g.,* personal addresses and public IPs). These operations are primarily scan-based, on which columnar storage formats like Parquet have significant advantages. Typically, HuggingFace Hub, the mainstream dataset hosting platform, adopts Apache Parquet format (Apache, 2023). Parquet is an open-sourced standard columnar data format, useful for cloud storage and online analysis (Lhoest et al., 2021; HuggingFace, 2024c). In this paper, Parquet is the default columnar format.

Features. With a *block-based design*, Parquet is optimized for distributed storage systems like GFS (Ghemawat et al., 2003) and HDFS (Shvachko et al., 2010). Figure 2 illustrates Parquet's logical representation and physical layout. The Parquet format has a three-layer storage hierarchy: (1) *Row group* consists of logical data blocks that are visible to typical Parquet reading APIs. (2) *Columnar chunk.* Each row group consists of multiple columnar chunks, e.g., chunk `x0` and chunk `y0` in Row Group 0, shown in Figure 2(a). Each Parquet I/O accesses a columnar chunk of a row group. (3) *Data page.* Each columnar chunk is divided into pages, which are the units of data compression. Pages within the same chunk (e.g., page 0 and page 1 of `x0` in Figure 2(b)) are decompressed in parallel.

3 LLM Data Pipeline Requirements

To guarantee the efficiency and accuracy of model training, a modern LLM training data pipeline has four important requirements: (1) enough capacity in storage systems, (2) alleviation of memory contention, (3) sufficient I/O throughput, and (4) high shuffle quality.

Table 1. Comparison of different training data pipeline solutions.

Method	Mem. Efficiency	Tput.	Model Acc.
Dist. Memory	✗	✓	✓
Disk. MMap	✓	✗	✓
Streaming	✓	✓	✗
YOUMU	✓	✓	✓

3.1 Ever Increasing Storage Cost of LLM Datasets

Storing massive LLM datasets in near-GPU high performance storage systems has become increasingly expensive, with costs even comparable to GPU expenses when considering redundant data copies (Google, 2024b). This escalating storage cost is primarily driven by the following two factors.

Dataset format conversion. Due to the lack of efficient row-wise random access in Parquet, data often needs to be converted into other formats before being fed into the training pipeline. We observe that for LLM datasets conversion from Parquet to JSON requires around 3× more storage space. This conversion process not only results in terabytes of redundant data but also complicates data management, breaking the rule of "single source of truth for the data"[1].

Frequent dataset update. Data preparation pipelines are continuously refined based on user feedback from model inference. Dynamic datasets often requires more up-to-date data and broader domains of knowledge. Companies frequently upgrade their models, releasing new versions approximately every three months (OpenAI, 2024; Anthropic, 2024). Each update mandates changes to the training dataset, leading to continuous format conversion overheads over time. In the long run, format conversion is a costly pitstop between data preparation and LLM training, while not acquiring enough attention.

3.2 Extensive Memory Demands of Training Systems

The consumption of memory footprint is performance-critical in model training frameworks. In our context, it is important to keep the memory footprint consumed by the data pipeline as low as possible, because there are other memory-intensive functionalities contending memory resources with the data pipeline.

Tensor offloading (Ren et al., 2021; Fang et al., 2023; Yuan et al., 2024) enables training of larger models and datasets beyond GPU memory capacity by dynamically transferring tensors between GPU memory and host DRAM. This technique often requires tens of gigabytes of DRAM per replicate, which accumulates to hundreds of gigabytes on a single node.

[1]Ensuring data consistency between formats of the dataset, each with multiple versions.

Model checkpointing (Mohan et al., 2021a; Eisenman et al., 2022; Wang et al., 2023) is crucial for rapid restoration of the training process in the event of system failures, with host DRAM serving as the fastest storage medium for these checkpoints. As models grow in size and more checkpoints are retained, DRAM requirements increase correspondingly. For instance, a single checkpoint of GPT-3 175B (Brown et al., 2020) model occupies $\sim$2.3 TB DRAM, reaching the limit of typical system DRAM capacities.

3.3 Sufficient I/O Throughput for GPU Utilization

To maximize GPU utilization in LLM training, it is common practice to overlap data loading with the previous iteration's training, effectively hiding the data loading latency. However, this strategy has limitations when the data loading latency exceeds the iteration time, resulting in GPU utilization drops due to data waiting. With the rapid advance of GPU computing power, the training workload becomes more sensitive to the iteration batch latency (MLCommons, 2024). The diversity of training tasks and cluster configurations necessitates a data loading pipeline with sufficiently high throughput to accommodate various scenarios.

3.4 High Shuffle Quality for Model Accuracy

Data shuffling is a necessary step for guaranteeing high model accuracy. However, better shuffle quality with larger datasets comes alongside higher shuffling overheads, which can be prohibitive to the overall training performance (Yang & Cong, 2019; Kumar & Sivathanu, 2020; Sun et al., 2022; Liu et al., 2023). Since the model sensitivity to shuffle quality varies, there are numerous pseudo-shuffle strategies to alleviate the high overheads of fully random shuffling, which results in the highest possible shuffle quality.

4 MOTIVATION

To avoid costly dataset conversions while keeping shuffling quality as high as possible, we need to (1) preserve as much randomness as possible in the shuffling process, and (2) directly shuffle datasets in columnar storage at low overheads. The former achieves as high shuffle quality as fully-random shuffle, and the latter realizes higher shuffling efficiency. Next, we show the following limitations we observe to motivate our designs.

4.1 Limitations of Existing Solutions

While some existing systems can already take data in columnar storage as input for LLM training (§3.1), none of them adequately satisfy all requirements in §3. We analyze the limitations of the state-of-the-art as follows.

Distributed memory (Rocklin, 2015; Moritz et al., 2018; Luan et al., 2023) loads and distributes data across multiple nodes in a cluster, leveraging the combined memory resources to handle massive datasets. Essentially, they cache the entire dataset in memory for fast data access. However, such a method correlates the memory footprint with dataset size. Given limited memory size, pre-loading LLM datasets at tens of terabytes can be infeasible. Moreover, the need for random access across the entire dataset necessaries frequent inter-node exchanging, resulting in potential contention with GPU communications (Dryden et al., 2021) and subsequently affects training speed.

Memory mapping from disk (Pumma et al., 2019; HuggingFace, 2024a) provides fast access to on-disk datasets, backed by a memory-mapped cache. This approach allows out-of-core processing on datasets larger than available physical memory. The users can randomly request any row from the dataset through a memory-mapped table which leverages virtual memory capabilities for fast lookups. While the OS page cache mechanism automatically manages data placement and eviction through demand paging, this approach can lead to performance degradation due to thrashing when access patterns trigger frequent page faults (Kumar & Sivathanu, 2020; Mohan et al., 2021b). Also, the memory overhead of the memory-mapped table itself can be high for large datasets. We notice that for a dataset with 200B tokens, one instance of its memory-mapped table takes around 12GB memory space and often each GPU needs one instance to keep track of data loading process.

Streaming is a widely used practice of pseudo-shuffle and it is one of the operations highly optimized in columnar storage format. It sequentially reads large dataset partitions into an in-memory buffer and then randomly samples data from the buffer (Hambardzumyan et al., 2022; TensorFlow, 2024a; HuggingFace, 2024b; PyTorch, 2024). This is often referred to as *local shuffle* as well. We find that using local shuffle in LLM training can degrade model accuracy significantly (§7).

4.2 Observations

Given the limitations of existing solutions, we aim to achieve memory efficiency, sufficient throughput and high shuffle quality for columnar storage-based LLM data pipeline simultaneously. Our insight is to *explicitly manage the disk I/O* instead of delegating them to the underlying implicit data movement by memory mapping or distributed memory. In this way, we aim to demonstrate that high performance storage systems are fast enough in fine-grained I/O for LLM training if their bandwidth can be sufficiently utilized. In detail, we have the following observations:

Marginal benefits of caching large datasets (O1). Although the data random access order during training can be predetermined, the data access to the same data point is exactly once per epoch. Since data use in one training step would not be reused again until the next epoch, the benefits of caching is minimal unless the entire dataset can fit in the cache (Zhu et al., 2018; Kumar & Sivathanu, 2020). Given the massive size of LLM datasets, full size can be challenging for the system memory size. Consequently, disk I/O remains an inevitable bottleneck.

Granularity gap between shuffling and I/O (O2). For columnar storage, the default I/O unit is a column chunk, typically hundreds of MB or even GB for an optimized columnar data reading setup. This implies that when requesting random rows, only around 0.1% bandwidth is effective. This leads to significant bandwidth waste. However, we notice that nowadays near-GPU storage systems are often backed by solid state drives (SSD) (Wei et al., 2023; Meta, 2024), which can tolerate page-level data access without degrading performance, suggesting an opportunity for finer-grained access (Jun et al., 2024).

5 YOUMU DESIGN

YOUMU is a practical high-performance data pipeline that is directly built on columnar storage in Parquet with fine-grained I/O and low memory footprint. Based on our observations (§4.2), YOUMU has the following design highlights:

No cache, only buffer (from observation **O1**). We focus on optimizing direct disk I/O access pattern and its efficiency, instead of trying to alleviate it through extensive caching. We only reserve a fixed size buffer for overlapping with training to hide I/O latency and in-memory shuffling to further enhance the shuffle quality when needed. In this way, YOUMU achieves very low memory footprint regarding very large datasets.

Fine-grained unified access (from observation **O2**). To provide high shuffle quality without caching, the disk I/O granularity needs to be fine-grained in the first place to avoid wasting I/O bandwidth due to the granularity gap issue. Ideally, the disk I/O granularity should be matched with the granularity of shuffling operations for the highest goodput.

Practical compatibility. We build YOUMU on the existing widely adopted columnar storage format Parquet instead of inventing new ones. This makes YOUMU highly practical in real-world systems and compatible with existing data processing ecosystems.

5.1 System Architecture

The overview of YOUMU is shown in Figure 3. YOUMU directly works with Parquet files on the storage and its metadata without any modifications. The upper-level training frameworks can call YOUMU Python APIs for integration

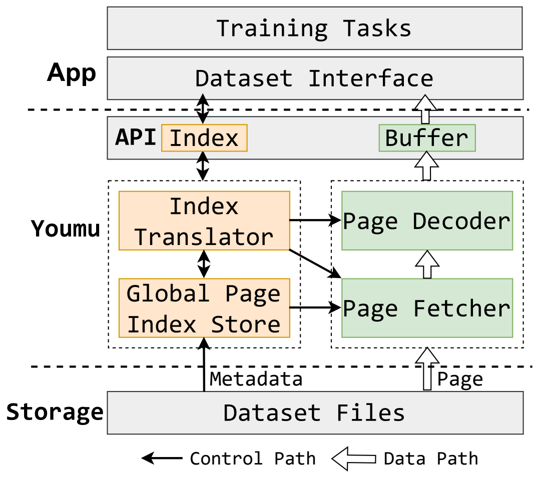

Figure 3. YOUMU system overview.

with their provided data interface (*e.g.,* PyTorch Dataset). In essence, YOUMU serves as an I/O backend to datasets stored as Parquet files and requires minimal modifications to existing systems for adoption.

YOUMU provides page-level data granularity (§5.2) for fine-grained data access and shuffling. The control plane (§5.3) collects Parquet metadata to build *global page index*. With the global page index and the page index translator, YOUMU can obtain the physical address of a random page and the necessary metadata for decoding. With given data location and metadata, the data plane (§5.4) can directly retrieve the page data from the disk, without the need of caching the datasets in memory or memory mapping. The extreme flexibility and compatibility of YOUMU also enables user-defined pipeline for their custom last-mile processing and aggressive buffer shuffling for enhanced shuffle quality.

5.2 Page-level Data Granularity

YOUMU implements page-level data access and shuffling as its core mechanism. We justify this design choice from the following two aspects:

Improving I/O performance. Since modern SSD are optimized for random access at page level (recall §4.2 **O2**), we can leverage most of the storage device performance by reading random pages. Additionally, by direclty reading pages inside the data column of interest, YOUMU naturally skips columns that are not in use. This saves I/O bandwidth and avoids runtime overheads of alignment between columns. The saving comes from the fact that only one column is needed for most data-intensive auto-regressive LLM training including pretraining and continual training.

Improving shuffle quality. Page is the minimal divisible data storage unit in Parquet since it is the level at which data is compressed. Considering that a typical page size is at multiple KB or MB, a large LLM dataset at TB-level would contain many millions of pages, which provides much finer-grained shuffle than simply shuffling the order of several hundred files or column chunks. Previous work (Zhu et al., 2018; Xu et al., 2022) has shown that the shuffle quality is heavily impacted by shuffle unit size.

Challenges. The `PageIndex` structure within the Parquet file metadata contains the page data's physical locations. However, the one-dimensional index of page physical locations alone is insufficient to achieve random page level data access. The page data requires decoding based on metadata defined in higher storage hierarchies. Moreover, the number of pages in each column chunk and file is uneven, which prevents the direct identification of belonging column chunk and file for an arbitrary page. Hence, *we need to build a mapping between page data address and their corresponding decoding-related metadata in higher-level hierarchies*. We address this challenge by introducing *Global Page Index* and its index translation mechanism to navigate through multiple storage hierarchies and metadata retrieval, which will be introduced next.

5.3 Control Plane with Global Page Index

Lightweight initialization. At initialization, YOUMU collects metadata from column chunks, which includes the physical addresses of pages within these chunks by `PageIndex`. While the native page index of columnar formats is typically limited to individual column chunks, YOUMU constructs a *global page address index* that spans the entire dataset. This index is structured as a three-dimensional matrix, incorporating file index, column chunk index, and page index. This comprehensive indexing approach enables efficient navigation and access across the full scope of the dataset. At the same time, the file-level and column chunk-level page number offset lists are created for the convenience of index translating.

Index translation. To determine the random page access order, we permute an integer list sized to the total page count. This random order is then translated to physical page addresses using a *layered binary search* as it is outlined in Algorithm 1. It first identifies the file containing the requested page through a binary search of file-level page number offsets. Then it locates the specific column chunk within the identified file via binary search of column chunk-level page number offsets. Finally it calculates the page index within the column chunk by subtracting the chunk page number offset from the random access order ID. This approach of establishing a global page index and performing efficient index translation at runtime allows YOUMU to avoid full materialization of the dataset in system memory for shuffling.

Metadata mapping. After we have the file, column chunk

Algorithm 1 Layered Binary Search-Index Translation

```
Input: Random page access order $id$, File-level page offset list $F$, Column chunk-level page offset list $C$
Output: Three-dimensional index for global page index
$fileIdx \leftarrow BinSearch(F, id)$
$chunkIdx \leftarrow BinSearch(C[fileIdx], id)$
$pageIdx \leftarrow id - C[fileIdx][chunkIdx]$
```

and page index for a random page access ID, we consult the global page index to retrieve the corresponding metadata and page physical location. To avoid incurring additional disk I/O for metadata, the global page index caches all the metadata and page locations in memory at initialization.

5.4 Data Plane with Page-level I/O

In this section, we detail the step-by-step process by which YOUMU's data plane retrieves and decodes the requested data from storage, utilizing the page location and metadata provided by the control plane.

Page data retrieval. Given the physical address range of a page, we directly extract the specified data segment from the file and load it into memory. Since Parquet pages are column-specific (each page contains data for only one column, as it is a sub-unit of a column chunk), unrelated columns are naturally bypassed, thereby conserving disk I/O bandwidth.

Decoding. Once the requested page is loaded into DRAM, it remains in a compressed state. Decompression requires metadata from the column chunk to which the page belongs, as provided by the control plane. This metadata includes crucial information about the data type and the specific decompression algorithm needed. After decompression, YOUMU constructs an in-memory page reader to convert the page data into Arrow arrays.

Aggressive buffer shuffle. Due to the page-level I/O granularity, the data retrieval unit size in YOUMU is smaller than the typical buffer size, presenting opportunities to enhance shuffling quality. Unlike standard double buffering which typically shuffles only once at initialization, YOUMU aggressively reads new pages into the buffer as soon as possible and performs shuffling upon the arrival of each new page. As outlined in Algorithm 2, YOUMU estimates an approximate row count per page (`#rowPage`) during initialization and refills the buffer with a new page when more than `#rowPage` rows have been consumed. This approach results in superior buffer shuffle quality compared to standard buffering. By complementing page-level data access shuffling, YOUMU achieves exceptionally high shuffle quality for data loading.

User-defined last-mile preprocessing. YOUMU utilizes Ar-

Algorithm 2 Aggressive Buffer Shuffle

```
Input: Dataset $D$, Buffer size $B$, Number of rows per page $\#rowPage$, Batch size $BS$
Output: Shuffled data in buffer
Initialize an empty buffer $bf$
{Initial buffer filling}
while size$(bf) < B$ do
    $page \leftarrow$ ReadRandomPage$(D)$
    Append$(bf,$ Decode$(page))$
end while
{Continuous refilling and shuffling}
while true do
    while size$(bf) + \#rowPage \leq B$ do
        $page \leftarrow$ ReadRandomPage$(D)$
        Append$(bf,$ Decode$(page))$
        ShuffleRows$(bf)$
    end while
    batch $\leftarrow bf$.pop$(BS)$
end while
```

Table 2. A partial list of YOUMU Python APIs.

Function	Parameters
`get_idx_matrix`	`file_paths, col_id`
`read_page`	`global_page_idx`
`shuffle_index`	`index_matrix`

row arrays as a versatile in-memory model. This approach facilitates seamless *zero-copy* conversion to dataframe formats (Wes McKinney, 2010; Harris et al., 2020), as well as tensor formats used in deep learning frameworks (Abadi et al., 2015; Chen et al., 2015; Paszke et al., 2019) via DLPack (DLPack contributers, 2024).

6 IMPLEMENTATION

We implement YOUMU on top of Apache Parquet and Arrow's official Rust implementation, utilizing `PyO3` to provide Python bindings. In this way, YOUMU plays as the replacement of I/O modules from PyArrow but still results in PyArrow objects for wide compatibility.

Rust runtime. Our implementation adds a new module `direct_page.rs` that encapsulates all functionality while maintaining compatibility with existing APIs. To enable direct page access, we leverage experimental features of Arrow readers and expose specific internal states to access page physical types and decompression utilities. The core functionality of YOUMU is efficiently implemented in approximately 500 lines of Rust code.

Python APIs. We leverage `arrow-rs::pyarrow` and `PyO3` to seamlessly integrate with Python, returning PyArrow objects from the Rust runtime. As shown in Table 2, we expose low-level page access APIs that enable users to implement custom logic compatible with standard interfaces such as PyTorch Dataset. In our PyTorch implementation, `get_idx_matrix` is invoked at dataset initialization, with the resulting list being shuffled later by `shuffle_index` and distributed across workers for non-overlapping random page access. When reading data from disk, `read_page` is used with given requested global page index. This entire PyTorch Dataset implementation requires only approximately 100 lines of Python code. We present an example of building PyTorch dataset class with YOUMU Python APIs in listing 1.

Listing 1. Youmu Dataset Interface Example

```python
import torch
import numpy as np
from torch.utils.data import IterableDataset
from youmu import get_idx_matrix, read_page, shuffle_index

# this is a comment
class YoumuDataset(IterableDataset):
    def __init__(...):
        self.index_matrix = get_idx_matrix(dataset_path, col_id)
        self.shuffled_idx_matrix = shuffle_index(self.
            index_matrix)
        ... # other attributes
    def __len__(self):
        # can be inferred from Parquet metadata
        return self.total_row_num
    def fill_buffer(self):
        while len(self.buffer) < self.buffer_size:
            new_samples = self.read_page(self.
                shuffled_index_matrix.pop(0))
            tensor_array = torch.from_dlpack(new_samples)
            # can also extend buffer with a random row instead
            # to achieve row-wise shuffling
            self.buffer.extend(new_samples)
        np.random.shuffle(self.buffer)

    def __iter__(self):
        ... # logic for DataLoader multi-processing
        for _ in range(iter_start, iter_end):
            if len(self.buffer) < self.refill_threshold:
                self.fill_buffer()
            sample = self.buffer.pop(0)
            yield sample
```

Fully random shuffle support. YOUMU's fine-grained control over page I/O enables row-wise full shuffling. Fully random shuffle is implemented by extracting individual rows from randomly accessed pages while maintaining epoch integrity through tracking of sampled page and inner row index pairs. While this approach incurs some I/O waste from unused rows in fetched pages, it significantly reduces waste compared to traditional chunk-based I/O methods. Users can choose this method when I/O bandwidth is sufficient to accommodate such overhead.

7 EVALUATION: MODEL ACCURACY

In this section, we answer the question: how does YOUMU's page-level shuffle perform in terms of training accuracy? Our results show that it achieves model accuracy comparable to fully random shuffling, significantly outperforming the streaming-based local shuffle. This demonstrates the importance of high shuffle quality for LLM datasets and the effectiveness of YOUMU's page-level shuffle.

Table 3. The model perplexity results for model pre-training and continual training. Lower is better.

Shuffle method	Small LLaMa pre-training			OpenLLaMa-3B continual training
	WikiText-103 line-level	doc-level	arXiv	algebraic-stack
Row (Ideal)	12.671	12.718	12.316	6.318
Page (YOUMU 1MB)	**12.884 (+0.213)**	**12.874 (+0.156)**	**12.323 (+0.007)**	**5.071 (-1.247)**
Streaming (Baseline)	13.371 (+0.700)	13.570 (+0.852)	12.655 (+0.339)	24.741 (+18.423)
No shuffle (Worst)	14.324 (+1.653)	14.127 (+1.409)	12.801 (+0.485)	285.424 (+279.106)

Table 4. The model configuration for pre-training.

Config.	Number
num_hidden_layer	12
num_attention_heads	12
head_dimension	64
vocab_size	30522
max_sequence_length	512

We measure the *achieved validation perplexity*, a widely used metric in language model evaluation. We measure this metric when running two data-intensive tasks in LLM training over four distinct system approaches. The two tasks are *pre-training from scratch* and *domain-specific continual training*. The four system approaches are as follows.

Row-wise fully random shuffle: This delivers the ideal shuffle quality, serving as the upper bound for model performance in terms of accuracy (Meng et al., 2019). It provides perfect randomization but is often impractical for large datasets due to its high overheads.

Page-level shuffle: The shuffle strategy introduced by YOUMU, achieving both *efficient* data access and *high model accuracy* due to good shuffle quality. Note that YOUMU also supports fully random shuffle.

Streaming-based local shuffle: The most widely adopted shuffle strategy (HuggingFace, 2024b; TensorFlow, 2024a) for large datasets exceeding DRAM capacity. In all experiments, we set the same buffer size of 10K rows for both streaming and page-level shuffle.

No shuffle: This represents the worst possible shuffle quality, serving as the lower bound. It helps quantify the impact of shuffling on model performance.

7.1 Pre-training from Scratch

Task. Due to computational resource constraints, we were unable to conduct a full-scale large model pre-training. Instead, we constructed a language model that shares the same basic structure as the LLaMa model (Touvron et al., 2023), but with a total of 160 million parameters. This model serves as our subject for training from scratch. The detailed model configuration is presented in Table 4. The pre-training experiments has two folds: the first set is to train the model on small datasets until convergence and the second set is to train the model on a large dataset with given step numbers.

For the first fold, We conduct model pretraining on two widely used datasets: WikiText-103 (Merity et al., 2016) and arXiv abstracts. Note that for WikiText-103, we evaluate both doc-level and line-level organizations, meaning one row includes a full document or a single line of text, respectively, to show the robustness against different dataset organizations. For the second fold, we train the model on C4 English subset dataset for 120 thousand steps with two different page sizes, i.e., 10KB and 1MB, for YOUMU to show its robust effectiveness on larger datasets.

Results. As shown in TABLE 3, YOUMU achieves model perplexity results much closer to the perfect row-wise full shuffling, compared with the perplexity delivered by streaming-based shuffling. On average, when the model is trained to convergence with small datasets, YOUMU's approach results in only 10.6% of the perplexity degradation observed with no shuffling, compared to 53.3% for streaming-based shuffling.

For training results on a large dataset, the C4 English subset, both 10KB and 1MB configurations of YOUMU outperforms the streaming-based shuffle baseline. The YOUMU 10KB even slightly outperforms the ideal fully random shuffling, showing that YOUMU's page-level shuffling provides sufficient shuffle quality to ensure same level of model accuracy as the fully random shuffle. Regarding the explanation on outperforming fully random shuffle, our understanding here is that, the theory of "better shuffle randomness leads to better model accuracy" generally holds true but is not guaranteed in the real world. Meanwhile, "less shuffle randomness leads to poorer model accuracy" corroborates with our observations and strengthens the motivation of YOUMU.

7.2 Domain-specific Continual Training.

Task. We conduct continual training for the pre-trained OpenLLaMa-3B (Geng & Liu, 2023) on the algebraic-stack dataset (Azerbayev et al., 2023) for 5,000 steps.

Results. As shown in TABLE 3, YOUMU even outperforms the perfect row shuffling in this task. While row shuffling

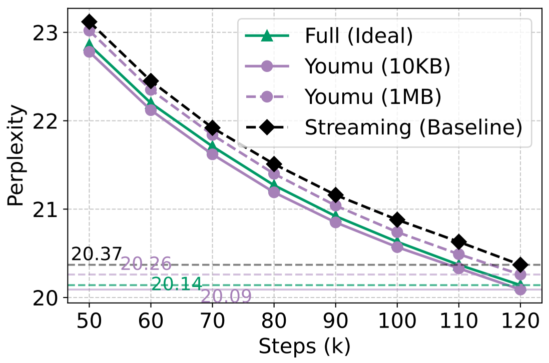

Figure 4. The validation perplexity curve of training with the C4 English dataset.

and YOUMU's perplexity results are good enough in practice, the streaming-based shuffling is still far from convergence. We think this is due to the variance of the data sampling window: while row and page shuffle can sample data from the entire dataset, the local and no shuffle can only see data from the very beginning window. This shows that YOUMU can greatly reduce the data amount requirement for domain-specific continual training, as constraining the data size is important to balance the model's capabilities in general and domain-specific fields (Ke et al., 2023), and prevent catastrophic forgetting (Li & Lee, 2024).

8 EVALUATION: SYSTEM EFFICIENCY

While §7 evaluates model accuracy through controlled-size full training, in this section we use methodologies in §8.1 to show how well YOUMU performs in terms of system efficiency when dealing with datasets with billion- or even trillion-level tokens. The key takeaways of the results are that across various settings: (1) Relying on a pipeline optimized for disk I/Os, YOUMU achieves comparable performance to the performance-optimal state-of-the-art, distributed memory-based solutions, and YOUMU significantly reduces memory footprint. (2) When sacrificing bandwidth for fully random shuffle, YOUMU still achieves sufficient performance with the least memory footprint.

8.1 Methodology

Baselines. From the system aspects, we choose the most representative systems as the baselines:

HuggingFace Datasets, a widely-used dataset library that implements the out-of-core data loading by disk-backed memory mapped file I/O mechanism. Although HuggingFace stores and downloads datasets in Parquet, at runtime it needs to convert them into Apache Feather format to enable memory mapped I/O.

Ray Data. a data processing library for ML workloads, backed by state-of-the-art distributed computing framework Ray with shared memory object store. We use a nightly built version of Ray to enable the latest push-based shuffle (Luan et al., 2023) for its best performance. Besides fully random shuffle, we also evaluate its chunk shuffle mode which has reduced shuffle overheads. We adopt the `TorchTrainer` class in Ray Train library for its implementation for distributed training. Note that the quality of such a chunk shuffle is still worse than YOUMU's page shuffle since its chunk size is often at several hundred MB, 1000 times larger than YOUMU's pages, indicating less potential shuffle randomness.

Dataset. We employ the cleaned English C4 (Raffel et al., 2020) dataset, which contains around 200B tokens and is widely used in moderate-size language model training. Also, we employ the latest 7 dumps of FineWeb dataset (Penedo et al., 2024) to enlarge the workload size, which contains 1.5T tokens. To study the isolated I/O performance, we run standard preprocessing and tokenization beforehand.

Testbed. 16 nodes in total are used in the evaluation. Each node has 384GB DRAM (150GB for shared memory) and is inter-connected by 100 Gb/s InfiniBand. The remote storage system is WekaFS (WekaIO, 2023). On each node, we run 8 concurrent data loading processes to simulate the case of 8 GPUs per node. The local batch size for each process is 64.

Experiment setup. We conduct two sets of experiments to show various aspects of the evaluated systems:

Fix workload, scaling workers. We study the scalability of evaluated systems by increasing involved nodes against the English C4 dataset. We name them as scalability tests.

Fix workers, scaling workload. We stress-test the system performance and overhead management by involving more partitions from the FineWeb dataset against a fixed number of workers. We denote them as stress tests.

Note that we need to exclude HuggingFace from the stress tests because the conversion to Apache Feather format requires significant extra storage space that is beyond our testbed capacity. For example, the C4 dataset employed in scalability tests only requires 500 GB in Parquet but requires 2.6 TB in Feather.

8.2 Memory Footprint & Disk Usage

Memory footprint. We present memory footprint for scalability tests in Figure 5 and stress tests in Figure 6, where we can see that YOUMU achieves the least memory footprint across all the tests, at most 82% and 76% less than HuggingFace and Ray Data respectively. Specifically, since YOUMU only holds dataset metadata and a fixed size buffer in memory, YOUMU only requires 30 GB for a 200B dataset,

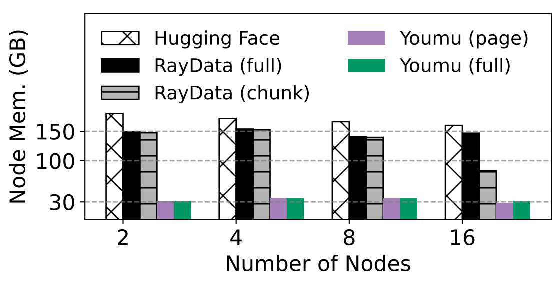

Figure 5. Scalability tests. Memory footprint per node against total node numbers with 200B token datasets.

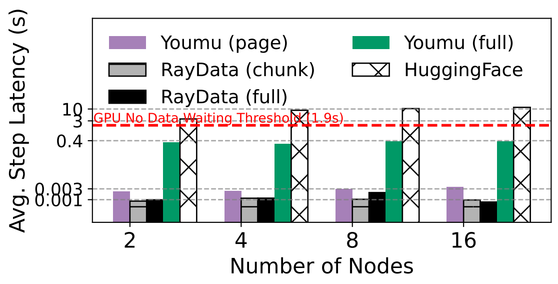

Figure 7. Strong scaling test. Batch latency against node numbers with the 200B token dataset.

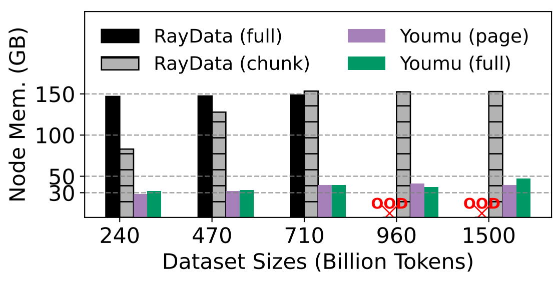

Figure 6. Stress tests. Memory footprint per node with various dataset sizes on 16 nodes.

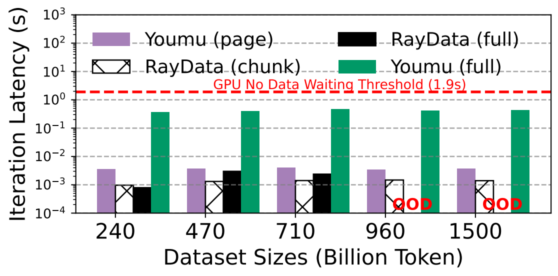

Figure 8. The step latency after Ray Data finishes shuffle wall.

and slightly increases to 47 GB at most for larger datasets for caching more metadata.

Disk usage by memory offloading. Ray Data employs the object spilling technique which offloads in-memory object store to disk when shared memory is not enough to fit the dataset. In such cases, disk I/Os would also be heavily involved at training runtime to fetch the data, which necessitates that the offloading space be a high-performance storage system located near the GPU. The size of the spilled object is related to shuffle granularity, number of workers, and dataset size. In Figure 6, Ray Data encounters out-of-disk errors with datasets larger than 1 trillion tokens (2 TB in Parquet) when doing full shuffling. We observe that in such cases it would need to write more than 10 TB data onto disk which triggered system kill.

8.3 Iteration Batch Latency

For the training data pipeline, the most important performance metric is *batch latency*, which is defined as the time needed to get a batch for the next training iteration. In practice we can overlap batch latency with current training iteration. Hence, as long as the batch latency is shorter than the time needed for one training iteration, the GPU utilization is not bottlenecked by data loading.

Batch latency threshold for GPU utilization. For language models, the training iteration step time can be estimated (Narayanan et al., 2021; Hoffmann et al., 2022) with given model parameter size, sequence length, batch size and GPU specifications. Typically such iteration step time for language models varies from *sub-second to tens of seconds* for various model and cluster sizes.

We present the iteration batch latency in Figures 7 and 8, with a threshold line at 1.9s indicating when the GPU experiences no data waiting time, a value we measured during real training in our previous model accuracy evaluation experiments. From this perspective, all the systems except HuggingFace deliver sufficiently low batch latency. YOUMU with page shuffling only has at most 3ms for batch latency. Although higher than Ray Data's 1ms, this is still three orders of magnitude lower than the data waiting threshold. With full shuffling, YOUMU has 0.4s batch latency, which is still acceptable for most training workloads. This is reasonable as YOUMU sacrifices goodput when using full shuffling.

Shuffle wall time per epoch. At the beginning of each epoch, Ray Data has an additional shuffle wall time due to its MapReduce-style shuffling. Since this period cannot be overlapped, the entire GPU cluster is idle during this stage. We illustrate Ray Data's shuffle wall time in Figure 9. In contrast, YOUMU only adds less than 10 seconds of metadata collection time in the first epoch.

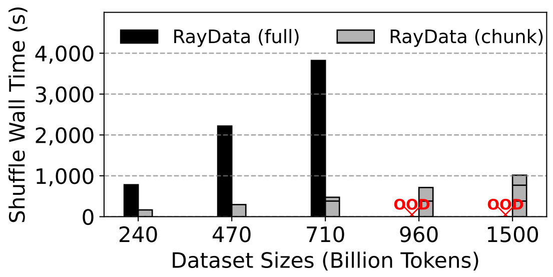

Figure 9. The shuffle wall time required by Ray Data on 16 nodes.

8.4 Overhead Discussion

Cross-language interface overheads. The `PyO3` interface introduces minimal performance overhead compared to a pure Rust implementation. The interface incurs a latency of nanoseconds per function call, primarily due to safety mechanisms such as GIL management and FFI boundary handling (Hewitt, 2021). For our scenarios, this negligible overhead does not materially impact performance.

CPU overheads. The system architecture of YOUMU implements a one-to-one mapping between GPUs and CPU processes, where each GPU is assigned a dedicated process that manages both I/O operations and decoding tasks. This design choice results in CPU core occupation equal to the number of GPUs . Since modern GPU computing nodes typically feature tens of CPU cores, the CPU resource overhead imposed by YOUMU remains within reasonable bounds and does not significantly impact overall system resources

9 Related Work

General shuffle services are very common in MapReduce systems (Dean & Ghemawat, 2008; Zaharia et al., 2012; Rasmussen et al., 2012) and sophisticated optimized to reorganize data between nodes or processes for query execution performance (Zhang et al., 2018; Pu et al., 2019; Shen et al., 2020). As predecessors of Ray Data, they rely on extensive memory resources and incur unavoidable shuffle wall time.

In-database machine learning systems aim to perform machine learning directly on database data using SQL expressions. While some systems support page-level granularity (Xu et al., 2022) for data stored in DB internal formats, they typically require implementing an entire deep learning stack inside database systems, isolating them from mainstream frameworks like PyTorch and TensorFlow. Furthermore, many of these systems focus on row-oriented database formats rather than the columnar formats prevalent in LLM dataset storage.

Storage formats for deep learning are specifically designed for training I/O pipeline acceleration (Aizman et al., 2019; Leclerc et al., 2023; TensorFlow, 2024b). Lance (LanceDB, 2024) is the most relevant format, which also targets efficient random access to columnar datasets. However, these formats often have incomplete functionality and their acceptance in real-world scenarios remains uncertain due to limited ecosystem support and compatibility.

Data loading services for deep learning encompass various optimization strategies, often tailored to specific architectures or tasks. These include shared dataset caching for multi-user scenarios (Kumar & Sivathanu, 2020; Gu et al., 2022), storage-hierarchy-aware prefetching schedules (Dryden et al., 2021), preprocessing acceleration (Graur et al., 2024; NVIDIA, 2024), and task-specific shuffle strategies (Sun et al., 2022; Nguyen et al., 2022). These techniques can integrate YOUMU as the I/O backend when dealing with LLM datasets in Parquet. Petastorm (Gruener et al., 2018) is most closely related to YOUMU, as it also aims to directly train from Parquet files. However, Petastorm operates at default row group-level I/O granularity, which leads to the aforementioned granularity mismatch. To avoid I/O waste, it loads entire row groups into the shuffle buffer, but this approach significantly compromises shuffle quality compared to YOUMU's page-level approach, as row groups are typically at GBs, thousands of times larger than pages.

10 Conclusions

This work presents YOUMU, a data pipeline that enables direct fine-grained columnar data feeding to GPUs for training on massive LLM datasets. We identify performance-prohibitive bandwidth waste due to granularity mismatch between fine-grained shuffling and columnar default chunk-based I/Os. YOUMU addresses the problem by re-designing columnar I/Os to be at page-level, and correspondingly performing page-level shuffling with sufficient randomness and thus achieving high model accuracy. Besides maintaining high shuffle quality and throughput, YOUMU also significantly reduces memory footprint.

Acknowledgments

We would like to express our sincere gratitude to the anonymous reviewers for their valuable feedback and constructive comments, which significantly improved the quality of this manuscript. This research was supported by the U.S. Department of Energy (DOE), Office of Advanced Scientific Computing Research (ASCR), under Award Number DE-SC0023452.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Aizman, A., Maltby, G., and Breuel, T. High performance i/o for large scale deep learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pp. 5965–5967, 2019. doi: 10.1109/BigData47090.2019.9005703.

Anthropic. Continuous model upgrades, 2024. URL https://docs.anthropic.com/en/docs/resources/model-deprecations. Accessed on 2024-10-30.

Apache. Apache parquet. https://github.com/apache/parquet-format, 2023.

Armenatzoglou, N., Basu, S., Bhanoori, N., Cai, M., Chainani, N., Chinta, K., Govindaraju, V., Green, T. J., Gupta, M., Hillig, S., Hotinger, E., Leshinksy, Y., Liang, J., McCreedy, M., Nagel, F., Pandis, I., Parchas, P., Pathak, R., Polychroniou, O., Rahman, F., Saxena, G., Soundararajan, G., Subramanian, S., and Terry, D. Amazon redshift re-invented. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, pp. 2205–2217, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3526045. URL https://doi.org/10.1145/3514221.3526045.

Azerbayev, Z., Schoelkopf, H., Paster, K., Santos, M. D., McAleer, S., Jiang, A. Q., Deng, J., Biderman, S., and Welleck, S. Llemma: An open language model for mathematics, 2023.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, 2015. URL https://arxiv.org/abs/1512.01274.

Choi, J., Kim, J., and Han, H. Efficient memory mapped file I/O for In-Memory file systems. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, Santa Clara, CA, July 2017. USENIX Association. URL https://www.usenix.org/conference/hotstorage17/program/presentation/choi.

Crawl, C. Common crawl dataset. https://commoncrawl.github.io/cc-crawl-statistics/, 2023.

Dean, J. and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51 (1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL https://doi.org/10.1145/1327452.1327492.

DLPack contributers. DLPack: an open in-memory tensor structure. https://dmlc.github.io/dlpack/latest/#, 2024.

Dryden, N., Böhringer, R., Ben-Nun, T., and Hoefler, T. Clairvoyant prefetching for distributed machine learning i/o. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3476181. URL https://doi.org/10.1145/3458817.3476181.

Eisenman, A., Matam, K. K., Ingram, S., Mudigere, D., Krishnamoorthi, R., Nair, K., Smelyanskiy, M., and Annavaram, M. Check-N-Run: a checkpointing system for training deep learning recommendation models. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 929–943, Renton, WA, April 2022. USENIX Association. ISBN 978-1-939133-27-4. URL https://www.usenix.org/conference/nsdi22/presentation/eisenman.

Fang, J., Zhu, Z., Li, S., Su, H., Yu, Y., Zhou, J., and You, Y. Parallel training of pre-trained models via chunk-based dynamic memory management. *IEEE Transactions on Parallel and Distributed Systems*, 34(1):304–315, 2023. doi: 10.1109/TPDS.2022.3219819.

Geng, X. and Liu, H. Openllama: An open reproduction of llama, May 2023. URL https://github.com/openlm-research/open_llama.

Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pp. 29–43, New York, NY, USA, 2003. ACM. ISBN 1-58113-757-5. doi: 10.1145/945445.945450. URL http://doi.acm.org/10.1145/945445.945450.

Google. Google bigquery. https://cloud.google.com/bigquery, 2024a.

Google. How to define a storage infrastructure for ai and analytical workloads. https://cloud.withgoogle.com/next/playlists?session=ARC306, 2024b.

Gorbunov, E., Hanzely, F., and Richtarik, P. A unified theory of sgd: Variance reduction, sampling, quantization and coordinate descent. In Chiappa, S. and Calandra, R. (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 680–690. PMLR, 26–28 Aug 2020. URL https://proceedings.mlr.press/v108/gorbunov20a.html.

Graur, D., Mraz, O., Li, M., Pourghannad, S., Thekkath, C. A., and Klimovic, A. Pecan: Cost-Efficient ML data preprocessing with automatic transformation ordering and hybrid placement. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 649–665, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-41-0. URL https://www.usenix.org/conference/atc24/presentation/graur.

Gruener, R., Cheng, O., and Litvin, Y. Introducing petastorm: Uber atg's data access library for deep learning. https://eng.uber.com/petastorm/, September 2018.

Gu, R., Zhang, K., Xu, Z., Che, Y., Fan, B., Hou, H., Dai, H., Yi, L., Ding, Y., Chen, G., and Huang, Y. Fluid: Dataset abstraction and elastic acceleration for cloud-native deep learning training jobs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pp. 2182–2195, 2022. doi: 10.1109/ICDE53745.2022.00209.

Hambardzumyan, S., Tuli, A., Ghukasyan, L., Rahman, F., Topchyan, H., Isayan, D., McQuade, M., Harutyunyan, M., Hakobyan, T., Stranic, I., and Buniatyan, D. Deep lake: a lakehouse for deep learning, 2022.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Hewitt, D. PyO3 performance analysis: function overheads, 2021. URL https://github.com/PyO3/PyO3/issues/1607. GitHub issue.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

HuggingFace. Datasets: Memory mapping. https://huggingface.co/docs/datasets/en/about_arrow#memory-mapping, 2024a.

HuggingFace. Dataset streaming. https://huggingface.co/docs/datasets/en/stream, 2024b.

HuggingFace. Analyze a dataset on the hub, 2024c. URL https://huggingface.co/docs/dataset-viewer/en/analyze_data.

Jun, Y., Park, S., Kang, J.-U., Kim, S.-H., and Seo, E. We ain't afraid of no file fragmentation: Causes and prevention of its performance impact on modern flash SSDs. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, pp. 193–208, Santa Clara, CA, February 2024. USENIX Association. ISBN 978-1-939133-38-0. URL https://www.usenix.org/conference/fast24/presentation/jun.

Ke, Z., Shao, Y., Lin, H., Konishi, T., Kim, G., and Liu, B. Continual pre-training of language models, 2023. URL https://arxiv.org/abs/2302.03241.

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

LanceDB. Lance: modern columnar data format for ML. https://lancedb.github.io/lance/, 2024.

Leclerc, G., Ilyas, A., Engstrom, L., Park, S. M., Salman, H., and Mądry, A. Ffcv: Accelerating training by removing data bottlenecks. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12011–12020, 2023. doi: 10.1109/CVPR52729.2023.01156.

Lhoest, Q., Villanova del Moral, A., Jernite, Y., Thakur, A., von Platen, P., Patil, S., Chaumond, J., Drame, M., Plu, J., Tunstall, L., Davison, J., Šaško, M., Chhablani, G., Malik, B., Brandeis, S., Le Scao, T., Sanh, V., Xu, C., Patry, N., McMillan-Major, A., Schmid, P., Gugger, S., Delangue, C., Matussière, T., Debut, L., Bekman, S., Cistac, P., Goehringer, T., Mustar, V., Lagunas, F., Rush, A., and Wolf, T. Datasets: A community library for natural language processing. In Adel, H. and Shi, S. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-demo.21. URL https://aclanthology.org/2021.emnlp-demo.21.

Li, C.-A. and Lee, H.-Y. Examining forgetting in continual pre-training of aligned large language models, 2024. URL https://arxiv.org/abs/2401.03129.

Li, J., Su, Q., Yang, Y., Jiang, Y., Wang, C., and Xu, H. Adaptive gating in mixture-of-experts based language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3577–3587, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.217. URL https://aclanthology.org/2023.emnlp-main.217/.

Liu, J., Nicolae, B., and Li, D. Lobster: Load balance-aware i/o for distributed dnn training. In *Proceedings of the 51st International Conference on Parallel Processing*, ICPP '22, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450397339. doi: 10.1145/3545008.3545090. URL https://doi.org/10.1145/3545008.3545090.

Luan, F. S., Wang, S., Yagati, S., Kim, S., Lien, K., Ong, I., Hong, T., Cho, S., Liang, E., and Stoica, I. Exoshuffle: An extensible shuffle architecture. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, pp. 564–577, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702365. doi: 10.1145/3603269.3604848. URL https://doi.org/10.1145/3603269.3604848.

Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., and Vassilakis, T. Dremel: Interactive analysis of web-scale datasets. In *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pp. 330–339. VLDB Endowment, 2010. URL http://www.vldb2010.org/accept.htm.

Meng, Q., Chen, W., Wang, Y., Ma, Z.-M., and Liu, T.-Y. Convergence analysis of distributed stochastic gradient descent with shuffling. *Neurocomputing*, 337:46–57, 2019.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Meta. Building Meta's GenAI infrastructure, 3 2024. URL https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/.

Miao, X., Jia, Z., and Cui, B. Demystifying data management for large language models. In *Companion of the 2024 International Conference on Management of Data*, SIGMOD/PODS '24, pp. 547–555, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704222. doi: 10.1145/3626246.3654683. URL https://doi.org/10.1145/3626246.3654683.

MLCommons. New MLPerf storage v1.0 benchmark results show storage systems play a critical role in AI model training performance, 9 2024. URL https://mlcommons.org/2024/09/mlperf-storage-v1-0-benchmark-results/.

Mohan, J., Phanishayee, A., and Chidambaram, V. CheckFreq: Frequent, Fine-Grained DNN checkpointing. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pp. 203–216. USENIX Association, February 2021a. ISBN 978-1-939133-20-5. URL https://www.usenix.org/conference/fast21/presentation/mohan.

Mohan, J., Phanishayee, A., Raniwala, A., and Chidambaram, V. Analyzing and mitigating data stalls in dnn training. *Proc. VLDB Endow.*, 14(5):771–784, jan 2021b. ISSN 2150-8097. doi: 10.14778/3446095.3446100. URL https://doi.org/10.14778/3446095.3446100.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., and Stoica, I. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 561–577, Carlsbad, CA, October 2018. USENIX Association. ISBN 978-1-939133-08-3. URL https://www.usenix.org/conference/osdi18/presentation/moritz.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V. A., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., and Zaharia, M. Efficient large-scale language model training on gpu clusters using megatron-lm, 2021. URL https://arxiv.org/abs/2104.04473.

Nguyen, T. T., Trahay, F., Domke, J., Drozd, A., Vatai, E., Liao, J., Wahib, M., and Gerofi, B. Why globally re-shuffle? revisiting data shuffling in large scale deep learning. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1085–1096, 2022. doi: 10.1109/IPDPS53621.2022.00109.

NVIDIA. Nvidia data loading library (dali), 2024. URL https://developer.nvidia.com/DALI. Accessed: 2024-10-30.

OpenAI. Continuous model upgrades, 2024. URL https://platform.openai.com/docs/models/continuous-model-upgrades. Accessed on 2024-08-12.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Penedo, G., Malartic, Q., Hesslow, D., Cojocaru, R., Cappelli, A., Alobeidli, H., Pannier, B., Almazrouei, E., and Launay, J. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023. URL https://arxiv.org/abs/2306.01116.

Penedo, G., Kydlíček, H., allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., and Wolf, T. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Pu, Q., Venkataraman, S., and Stoica, I. Shuffling, fast and slow: scalable analytics on serverless infrastructure. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, pp. 193–206, USA, 2019. USENIX Association. ISBN 9781931971492.

Pumma, S., Si, M., Feng, W.-C., and Balaji, P. Scalable deep learning via i/o analysis and optimization. *ACM Trans. Parallel Comput.*, 6(2), July 2019. ISSN 2329-4949. doi: 10.1145/3331526. URL https://doi.org/10.1145/3331526.

PyTorch. Iterable style dataset. https://pytorch.org/docs/stable/data.html#iterable-style-datasets, 2024.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Rasmussen, A., Lam, V. T., Conley, M., Porter, G., Kapoor, R., and Vahdat, A. Themis: an i/o-efficient mapreduce. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450317610. doi: 10.1145/2391229.2391242. URL https://doi.org/10.1145/2391229.2391242.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. Zero-offload: Democratizing billion-scale model training, 2021. URL https://arxiv.org/abs/2101.06840.

Rocklin, M. Dask: Parallel computation with blocked algorithms and task scheduling. In *SciPy*, 2015. URL https://api.semanticscholar.org/CorpusID:63554230.

Shen, M., Zhou, Y., and Singh, C. Magnet: push-based shuffle service for large-scale data processing. *Proc. VLDB Endow.*, 13(12):3382–3395, August 2020. ISSN 2150-8097. doi: 10.14778/3415478.3415558. URL https://doi.org/10.14778/3415478.3415558.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–10. IEEE, 2010. doi: 10.1109/MSST.2010.5496972.

Sun, B., Yu, X., Zhang, C., Tian, J., Jin, S., Iskra, K., Zhou, T., Bicer, T., Beckman, P., and Tao, D. Solar: A highly optimized data loading framework for distributed training of cnn-based scientific surrogates, 2022. URL https://arxiv.org/abs/2211.00224.

Tang, L., Ranjan, N., Pangarkar, O., Liang, X., Wang, Z., An, L., Rao, B., Cheng, Z., Sun, S., Mu, C., Miller, V., Peng, Y., Liu, Z., and Xing, E. P. Txt360: A top-quality llm pre-training dataset requires the perfect blend, 2024.

TensorFlow. Tensorflow dataset api. https://www.tensorflow.org/api_docs/python/tf/data/Dataset, 2024a.

TensorFlow. Tfrecord example. https://www.tensorflow.org/tutorials/load_data/tfrecord, 2024b.

Together Computer. Redpajama: an open dataset for training large language models, October 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023.

Wang, Z., Jia, Z., Zheng, S., Zhang, Z., Fu, X., Ng, T. S. E., and Wang, Y. Gemini: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 364–381, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613145. URL https://doi.org/10.1145/3600006.3613145.

Wei, J., Zhang, X., Wang, L., and Wei, Z. Fastensor: Optimise the tensor i/o path from ssd to gpu for deep learning training. *ACM Trans. Archit. Code Optim.*, 20(4), December 2023. ISSN 1544-3566. doi: 10.1145/3630108. URL https://doi.org/10.1145/3630108.

WekaIO. WekaIO documentation. https://docs.weka.io/, 2023.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman (eds.), *Proceedings of the 9th Python in Science Conference*, pp. 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

Xu, L., Qiu, S., Yuan, B., Jiang, J., Renggli, C., Gan, S., Kara, K., Li, G., Liu, J., Wu, W., Ye, J., and Zhang, C. In-database machine learning with corgipile: Stochastic gradient descent without full data shuffle. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, pp. 1286–1300, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3526150. URL https://doi.org/10.1145/3514221.3526150.

Yang, C.-C. and Cong, G. Accelerating data loading in deep neural network training. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 235–245, 2019. doi: 10.1109/HiPC.2019.00037.

Yuan, T., Liu, Y., Ye, X., Zhang, S., Tan, J., Chen, B., Song, C., and Zhang, D. Accelerating the training of large language models using efficient activation rematerialization and optimal hybrid parallelism. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 545–561, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-41-0. URL https://www.usenix.org/conference/atc24/presentation/yuan.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient distributed datasets: A Fault-Tolerant abstraction for In-Memory cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 15–28, San Jose, CA, April 2012. USENIX Association. ISBN 978-931971-92-8. URL https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.

Zhang, H., Cho, B., Seyfe, E., Ching, A., and Freedman, M. J. Riffle: optimized shuffle service for large-scale data analytics. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355841. doi: 10.1145/3190508.3190534. URL https://doi.org/10.1145/3190508.3190534.

Zhong, T., Zhao, J., Guo, X., Su, Q., and Fox, G. Rinas: Training with dataset shuffling can be general and fast, 2023. URL https://arxiv.org/abs/2312.02368.

Zhu, Y., Chowdhury, F., Fu, H., Moody, A., Mohror, K., Sato, K., and Yu, W. 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)*, pp. 145–156, 2018. doi: 10.1109/MASCOTS.2018.00023.