Optimizing Data I/O for LLM Datasets on Remote Storage

Tianle Zhong
tianle.zhong@email.virginia.edu
University of Virginia

Jiechen Zhao
jc.zhao@mail.utoronto.ca
University of Toronto

Xindi Guo
ptk2ks@virginia.edu
University of Virginia

Qiang Su*
qiang.su@my.cityu.edu.hk
City University of Hong Kong

Geoffrey Fox*
vxj6mb@virginia.edu
University of Virginia

1 INTRODUCTION

Training large language models (LLMs) demands increasingly larger datasets for optimal performance [13]. In practice, these datasets may include hundreds of terabytes (TB) or even petabytes of data. For example, Common Crawl [3] needs approximately 7 petabytes of data and expands by 200TB to 300TB monthly. Even the deduplicated and cleaned open datasets like the RedPajama-data-v2 dataset [2] require 270TB of storage space.

While existing solutions like Ray [15, 19] have attempted to leverage distributed memory for efficient data pipelines in cluster-scale training, the capacity of system memory is limited at the O(1TB) level and cannot accommodate O(100TB) or even larger datasets for LLM training. Going down along the storage hierarchy, the local storage at rack-scale provides O(100TB) of storage space which looks hopeful. However, when considering the scalability, local storage becomes suboptimal due to their limited extensibility after deployment. By contrast, remote storage systems provide O(1PB) of storage space and comparable performance to local storage. Furthermore, remote storage systems can be easily extended and shared between clusters.

In this project, we showcase a study of the impact of *remote storage* for training language models on large datasets. We propose Reimu, an algorithm-system co-design approach, including (1) block-level shuffle as an approximation to the expensive full shuffle with minimal impact on model accuracy; (2) and a lightweight indexing method without the need to scan the whole dataset to optimize the I/O efficiency. Our preliminary results show that Reimu significantly optimizes the dataset initialization time and throughput, and Reimu has minimal impact on the model accuracy.

2 LLM TRAINING WITH REMOTE STORAGE

2.1 How Datasets are Stored and Read?

Dataset format in storage. We consider the common storage format in remote storage systems for web-scale datasets. *Column-oriented format* [6, 16] has been proposed for online analytical processing (OLAP) services like Google BigQuery [8] and Amazon Redshift [1] on web-scale datasets. The format introduces a *block-based design* that improves query processing and I/O performance by enabling efficient columnar compression and encoding schemes. This paper focuses on the column-oriented format as it has been the mainstream in use [14] and optimized for remote storage systems like GFS [7] and HDFS [23].

*Qiang Su and Geoffrey Fox are corresponding authors. The Univesity of Virginia team thanks NSF Grant 2210266 and DOE Grant DE-SC0023452 for partial support. We acknowledge the excellent work of the Rivanna HPC Cluster team.

Data reading order. In model training data pipelines, data shuffling is a necessary step to achieve better model accuracy and convergence, which has been proven in both practice and theorem foundation [17]. Shuffling satisfies the need for random data order for stochastic gradient descent-based learning [18]. There are two categories of shuffling strategies: (1) offline shuffling data order in the storage [5]; (2) and runtime changing the reading order [5, 24, 28]. In the context of very large datasets, the time cost and 2× storage space required by offline shuffling is unacceptable [5, 28]. Hence, in this work, we only discuss the runtime shuffling strategies.

2.2 I/O Efficiency Problems in LLM Training

In practice, the full shuffling is implemented as *random access without replacement* [9, 18]. For LLM datasets, this can be viewed as row-wise random access. However, column-oriented formats usually do not provide explicit row indices. A common practice is to leverage memory mapping and scan the entire dataset to create a mapping between virtual memory addresses and the file on disk. By using memory mapping, the operating system can load the required data pages into memory on-demand, even if the total dataset size exceeds the available RAM.

Slow initialization. The time needed for scanning the entire dataset to create the row indices is proportional to the dataset size. We estimate the initialization time on the English subset of Colossal Clean Crawled Corpus (C4) dataset [21] (referred to as C4-en in the following). This dataset is common to train large embedding models [10, 21]. After tokenization with a max sequence length of 512, the dataset has 386M rows and 2.4TB on disk. The initialization process takes around 5 minutes. This implies that it would take hours to days for datasets at the petabyte level. The dataset initialization time cost is non-trivial because, for large cluster-level training, the GPU failure becomes unavoidable and even frequent [4, 12, 26]. When restoring the training job, the model checkpoints need reloading, and the dataset needs reinitialization.

Page fault overheads. When the dataset size is much larger than the available RAM, accessing random rows of data can lead to frequent page faults and memory swapping. This happens because the operating system needs to make room for the requested data by paging out the least recently used memory pages to disk. When those pages are needed again, they must be loaded back into RAM, potentially causing further paging and swapping. This process can significantly degrade data I/O throughput and overall performance.

Wasted bandwidth. Moreover, due to the block-based design of the column-oriented format, the minimum reading unit is one block. For row-wise random access, only one row of each block is likely needed to generate the batch for feeding the model. Considering the common block size 1000, the application-level throughput (batch generation) is merely 0.1%.

Table 2 summarizes the performance impact of the aforementioned I/O efficiency factors on application-level throughput. We compare the batch generation throughput between row-wise random access and sequential reading. We can see that row-wise random access can be up to 47× slower than sequential reading. When using more GPUs to parallelize training, we expect that the slow data supply from remote storage due to the I/O inefficiency will lead to significant GPU idle time. This further makes it challenging for I/O overhead hiding techniques, such as prefetching, to work effectively, because I/Os cannot be fully overlapped by GPU computation.

2.3 Model Accuracy Degradation Problem

To avoid random I/O issues, a window-sliding shuffling method [24] is often used, where the dataset is streamed to a DRAM buffer and shuffled in-memory. Although this enables sequential reading and maximizes I/O performance, the window size is limited by the available buffer size. Given the size discrepancy between DRAM and remote storage, the buffer can be very constrained compared to the dataset size, leading to undesired model accuracy degradation.

3 REIMU: ALGORITHM-SYSTEM CO-DESIGN

In this section, we discuss how to train language models on large datasets with column-oriented formats with consideration of both I/O efficiency and model accuracy. Reimu has two design highlights: block-level shuffle and compute-to-index fetching.

Algorithm: Block-level shuffle. We introduce the block-level shuffle which shuffles the order of the blocks instead of rows. Our main idea is that one block is relatively very small considering the massive block number in total and still can be considered as an *approximation* to random access on the whole dataset. Furthermore, when dataset size (total block number) increases, the block-level shuffle becomes *more similar* to the row-wise full shuffle.

However, this may still compromise the shuffle quality compared to the row-wise full shuffling. To enhance the shuffle quality, after random blocks are read into the DRAM space, we further perform a in-memory row-wise shuffle as it is shown in Figure 1. This approach has been proven to be effective both formally and in practice for in-database machine learning [28] with relatively simple and small models and datasets. We share the same formal convergence analysis and in this work we further validate its effectiveness with training transformer-based text generation models. Different from [28], the block-based storage is the nature of column-oriented format so we do not need to manually divide datasets into blocks.

System: Compute-to-index. Instead of indexing data by memory mapping, we propose a compute-to-index approach to avoid the long initialization time, frequent page faults and memory swapping. Without memory mapping, we lost the convenient mapping from virtual addresses to storage space and have to figure out the data location at the runtime. Our main idea is that since the block size is fixed, we can infer the block position based on the block number of each dataset file. We first collect the block number of each file by reading their metadata and form a block number offset list against the whole dataset. When randomly indexing a data block, we perform a binary search on the block offset list to quickly determine which file contains the requested block and its local block index within that file. Once we have the file and local block index, we can directly access the required data block.

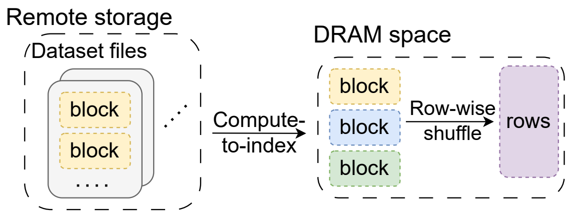

Figure 1: Reimu's datapath. Different blocks of different colors indicate that they are from different dataset files.

Strategy	Perplexity
Row-wise full shuffle	12.73
Reimu	**12.88 (+0.15)**
Window-sliding	13.36 (+0.63)
No shuffle	14.32 (+1.59)

Table 1: The converged tiny LLaMa model perplexity on the validation set of the wikitext-103 dataset with different loading strategies. Lower is better. Reimu and window-sliding share the same buffer size of 10,000 samples here.

Strategy	Init. time	Tput (samples/second)
Reimu	**~7 sec.**	**1893.7**
Row-wise full shuffle	~5 min.	44.4
Window-sliding	N/A	2069.9
No shuffle (ideal)	N/A	2075.3

Table 2: The initialization time and batch generation throughput comparisons for loading English subset of the C4 dataset. Batch size is set to 32.

It is worth noting that this approach can also be extended for row-wise random access. With given global row indices, we perform a two-stage binary search which first searches from files into blocks and then into rows.

4 EXPERIMENTS AND FUTURE WORK

Implementations & System settings. We prototype Reimu with PyArrow 14.0 and build the training pipeline with PyTorch 2.1. The dataset is stored on a remote storage system with WekaFS [27].

Workloads. We train a tiny LLaMa model [25] (160M) with different strategies until convergence on the wikitext-103 dataset. We measure the throughput of batch generation with the C4-en dataset.

Results. We report the converged model perplexity on the validation set in Table 1 and the batch generation throughput with a larger dataset C4-en in Table 2. First, Reimu successfully achieves similar model perplexity with the full shuffle. Second, when a larger dataset is employed, Reimu reduces initialization time from 5 minutes to 7 seconds and only introduces minimal computational overheads, keeping comparable throughput to window-sliding shuffle. Despite the limited model and dataset sizes used in this study due to computational constraints, we expect the convergence behavior to remain consistent across larger model and dataset pairs, based on the previous shuffle similarity analysis.

Future work. As a general software solution, we can integrate Reimu with dedicated hardware solutions such as NVIDIA GPUDirect storage [20] for further enhancement. To assess its gains in large-scale cluster training, we will integrate Reimu with Megatron-LM [22], benchmarking it across numerous GPUs and real datasets at the 100TB scale. Additionally, we're exploring Reimu's applicability in scenarios like cloud-based data reading [11, 14], where I/O bandwidth is lower, and incorporating on-the-fly tokenization to conserve storage while considering the computational costs.

REFERENCES

[1] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J. Green, Monish Gupta, Sebastian Hillig, Eric Hotinger, Yan Leshinksy, Jintian Liang, Michael McCreedy, Fabian Nagel, Ippokratis Pandis, Panos Parchas, Rahul Pathak, Orestis Polychroniou, Foyzur Rahman, Gaurav Saxena, Gokul Soundararajan, Sriram Subramanian, and Doug Terry. 2022. Amazon Redshift Re-invented. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2205–2217. https://doi.org/10.1145/3514221.3526045

[2] Together Computer. 2023. RedPajama: an Open Dataset for Training Large Language Models. https://github.com/togethercomputer/RedPajama-Data

[3] Common Crawl. 2023. Common Crawl Dataset. https://commoncrawl.github.io/cc-crawl-statistics/. Accessed: March 13, 2024.

[4] Assaf Eisenman, Kiran Kumar Matam, Steven Ingram, Dheevatsa Mudigere, Raghuraman Krishnamoorthi, Krishnakumar Nair, Misha Smelyanskiy, and Murali Annavaram. 2022. Check-N-Run: a Checkpointing System for Training Deep Learning Recommendation Models. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 929–943. https://www.usenix.org/conference/nsdi22/presentation/eisenman

[5] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. 2012. Towards a unified architecture for in-RDBMS analytics. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 325–336. https://doi.org/10.1145/2213836.2213874

[6] Apache Software Foundation. 2023. Apache Parquet. https://github.com/apache/parquet-format.

[7] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles* (Bolton Landing, NY, USA) *(SOSP '03)*. ACM, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450

[8] Google. 2024. Google BigQuery. https://cloud.google.com/bigquery. Accessed: 2024-03-17.

[9] Eduard Gorbunov, Filip Hanzely, and Peter Richtarik. 2020. A Unified Theory of SGD: Variance Reduction, Sampling, Quantization and Coordinate Descent. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 108)*, Silvia Chiappa and Roberto Calandra (Eds.). PMLR, 680–690. https://proceedings.mlr.press/v108/gorbunov20a.html

[10] Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, Maximilian Werk, Nan Wang, and Han Xiao. 2024. Jina Embeddings 2: 8192-Token General-Purpose Text Embeddings for Long Documents. arXiv:2310.19923 [cs.CL]

[11] Sasun Hambardzumyan, Abhinav Tuli, Levon Ghukasyan, Fariz Rahman, Hrant Topchyan, David Isayan, Mark McQuade, Mikayel Harutyunyan, Tatevik Hakobyan, Ivo Stranic, and Davit Buniatyan. 2022. Deep Lake: a Lakehouse for Deep Learning. arXiv:2209.10785 [cs.DC]

[12] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. 2024. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs. arXiv:2402.15627 [cs.LG]

[13] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv:2001.08361 [cs.LG]

[14] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A Community Library for Natural Language Processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Heike Adel and Shuming Shi (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 175–184. https://doi.org/10.18653/v1/2021.emnlp-demo.21

[15] Frank Sifei Luan, Stephanie Wang, Samyukta Yagati, Sean Kim, Kenneth Lien, Isaac Ong, Tony Hong, Sangbin Cho, Eric Liang, and Ion Stoica. 2023. Exoshuffle: An Extensible Shuffle Architecture. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 564–577. https://doi.org/10.1145/3603269.3604848

[16] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: Interactive Analysis of Web-Scale Datasets. In *Proc. of the 36th Int'l Conf on Very Large Data Bases*. VLDB Endowment, 330–339. http://www.vldb2010.org/accept.htm

[17] Qi Meng, Wei Chen, Yue Wang, Zhi-Ming Ma, and Tie-Yan Liu. 2019. Convergence analysis of distributed stochastic gradient descent with shuffling. *Neurocomputing* 337 (2019), 46–57.

[18] Konstantin Mishchenko, Ahmed Khaled, and Peter Richtárik. 2020. Random reshuffling: Simple analysis with vast improvements. *Advances in Neural Information Processing Systems* 2020-December (2020). https://www.scopus.com/inward/record.uri?eid=2-s2.0-85106134827&partnerID=40&md5=e08bcb7e24168b446b28aceec3f92b54 Cited by: 31.

[19] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 561–577. https://www.usenix.org/conference/osdi18/presentation/moritz

[20] NVIDIA. 2023. NVIDIA GPUDirect Storage. https://docs.nvidia.com/gpudirect-storage/index.html. Accessed: March 13, 2024.

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html

[22] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL]

[23] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–10. https://doi.org/10.1109/MSST.2010.5496972

[24] TensorFlow. 2024. TensorFlow Dataset API. https://www.tensorflow.org/api_docs/python/tf/data/Dataset. Accessed: March 13, 2024.

[25] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[26] Zhuang Wang, Zhen Jia, Shuai Zheng, Zhen Zhang, Xinwei Fu, T. S. Eugene Ng, and Yida Wang. 2023. GEMINI: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles* (<conf-loc>, <city>Koblenz</city>, <country>Germany</country>, </conf-loc>) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 364–381. https://doi.org/10.1145/3600006.3613145

[27] WekaIO. 2023. WekaIO Documentation. https://docs.weka.io/. Accessed: March 13, 2024.

[28] Lijie Xu, Shuang Qiu, Binhang Yuan, Jiawei Jiang, Cedric Renggli, Shaoduo Gan, Kaan Kara, Guoliang Li, Ji Liu, Wentao Wu, Jieping Ye, and Ce Zhang. 2022. In-Database Machine Learning with CorgiPile: Stochastic Gradient Descent without Full Data Shuffle. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1286–1300. https://doi.org/10.1145/3514221.3526150