Lotus: Characterize Architecture Level CPU-based Preprocessing in Machine Learning Pipelines

Rajveer Bachkaniwala
rr@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Harshith Lanka
hlanka3@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Kexin Rong
krong@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Ada Gavrilovska
ada@cc.gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Abstract

In machine learning (ML) pipelines, preprocessing tasks—such as loading, decoding, and applying transformations—require substantial compute and power resources. With the slowdown of Moore's Law, the benefits of traditional hardware scaling are diminishing, making the optimization of these preprocessing tasks increasingly critical for overall pipeline efficiency. While previous works have introduced various software optimizations to address the preprocessing bottleneck, less attention has been given to optimizing these tasks in relation to the underlying CPU architecture's efficiency. This is a missed opportunity, as the performance of preprocessing is closely tied to the CPU's microarchitecture, memory hierarchy, and instruction pipeline efficiency.

Our work addresses this gap by introducing Lotus, an open-source profiling tool specifically designed for the preprocessing stage of ML pipelines. Lotus supports future optimizations across the hardware-software stack, by providing insights that allow practitioners to evaluate the limitations of their CPU architecture in the context of preprocessing tasks, and assess the efficiency of their preprocessing pipelines under different configurations. The tool is available at https://github.com/rajveerb/lotus.

1 Introduction

Preprocessing is a critical step in machine learning (ML) pipelines, where raw input data is ingested and transformed into a format suitable for ML models. This step typically involves a chain of complex operations, such as loading, decoding, and applying transformations, which can demand substantial compute resources. For example, preprocessing can consume up to 65% of the epoch time in tasks such as image classification, object detection, and audio classification [38], and CPU power consumption during preprocessing can account for over 20% of the total power usage in certain ML workloads [51].

The 2nd Workshop on Hot Topics in System Infrastructure (HotInfra'24), co-located with SOSP'24, November 3, 2024, Austin, TX, USA

At the same time, efficient preprocessing is crucial for ML training jobs, which require low latency (100 μs - 1 ms) and high throughput (10 GB/s) for per-batch generation [50]. Inefficiencies in CPU-based preprocessing can cause low utilization of expensive accelerators, especially in systems with an imbalance between CPU and accelerator resources [12, 33, 38, 40, 41, 46].

Various optimizations from both academia and industry have been proposed to enhance preprocessing performance. These include parallelizing I/O and compute within and across batches [30, 39, 44], offloading preprocessing tasks to accelerators (DALI [9], TrainBox [42]), data duplication [16], caching optimizations [12, 22, 27, 31, 38, 44], dataset storage improvements [10, 31], disaggregated preprocessing across nodes [12, 22, 23, 47, 49, 50], and co-locating ML jobs for efficient caching and scheduling in clusters [32, 48, 49].

The range of these optimizations highlights the importance of the preprocessing stage in ML training pipelines. However, it has been consistently shown that significant efficiency gains can be achieved through better hardware design. Both cloud providers [2, 5, 8] and hardware vendors offer customizable infrastructure configurations for specific workloads, including options for different types of CPUs, GPUs/accelerators, and memory. Workload-specific hardware designs, such as new accelerators (e.g., TPUs [28], IPUs [7]) or workload-specialized CPUs (e.g., AWS Graviton [4], Microsoft's Cobalt 100 [36], Google's Axion [34]), have demonstrated both performance and system efficiency improvements. Additionally, vendors [15, 19, 25, 26, 37] offer AI/ML servers (e.g., NVIDIA HGX H100 and H200, AMD MI300X, Intel Gaudi2) with various SKUs for CPU, memory, SSD, and NIC configurations. Evaluating the limitations of the CPU SKU in AI/ML servers is crucial, as it directly impacts CPU-based preprocessing performance in ML training clusters.

A critical capability for designing or configuring workload-specific hardware is the ability to finely characterize workload resource requirements and identify performance and scalability bottlenecks within the existing infrastructure stack. However, our research reveals a lack of effective profiling tools that can adequately characterize the performance of preprocessing pipelines at the CPU architectural level.

In this paper, we describe Lotus, presented at IISWC 2024 [13], an open-source profiling tool specifically developed for the preprocessing stage of ML pipelines. Lotus combines lightweight instrumentation and tracing of ML preprocessing workloads with a novel method for approximating the mapping of hardware events to individual preprocessing operations. This provides detailed insights into the performance characteristics and microarchitectural bottlenecks of preprocessing workloads. These insights enable infrastructure designers to evaluate the limitations of their CPU architecture and the efficiency of their preprocessing pipelines across different ML training job configurations.

We present an illustrative example that demonstrates the utility of these insights by applying Lotus to a representative ML workload. We show that Lotus can capture fine-grained preprocessing timing, and get a CPU architectural-level performance view of their execution, which helps reveal a clear shift in the workload's performance bottlenecks under different job configurations. Our full paper [13] includes a detailed explanation of the design of Lotus, as well as an extended comparison of its capabilities, overheads, and ease of use with respect to alternative profilers including Scalene [14], py-spy [21], austin [45] and the PyTorch profiler [11].

2 Lotus Design

Challenges. The design of Lotus addresses two main challenges in existing solutions. First, there is a disconnect between the performance of high-level Python functions and low-level hardware metrics (such as L1 cache misses) that are collected via performance counters. Existing hardware profilers, such as Intel VTune [6] and AMD uProf [3], collect CPU cache and microarchitecture performance data for C/C++ functions but cannot capture stack frames of machine learning pipeline code written in Python. Additionally, Python profilers that capture the call stack often fail to label preprocessing functions correctly, forcing users to investigate the source code manually to recreate the stack trace.

Second, capturing fine-grained batch-level preprocessing timing data with low overhead is challenging. Sampling-based Python profilers such as Scalene [14], py-spy [21], and austin [45] are constrained by their sampling rates, making it challenging to capture the duration of individual transformation operations that may only take hundreds of microseconds to a few milliseconds without incurring significant overhead. Moreover, the asynchronous data flow used in many preprocessing frameworks, where worker processes execute the actual preprocessing operations while the main process coordinates, complicates the measurement of elapsed times. Recent work on optimizing preprocessing pipelines [30, 39, 44] rely on instrumentation to capture aggregated elapsed time across many batches, but does not capture fine-grained per-batch statistics or data flow dependencies.

To address these limitations, we make two key observations. First, ML preprocessing pipelines are often declaratively defined, providing hooks for fine-grained instrumentation while ensuring generalizability across different pipelines and frameworks [33, 39, 43]. Second, once such fine-grained instrumentation data is available, it can be leveraged to better attribute low-level hardware performance counters measured by the hardware profilers to the corresponding high-level preprocessing functions.

Design Overview. We leverage these insights to build **Lotus** – a new profiling tool for ML preprocessing pipelines declared using PyTorch's DataLoader [43].

Lotus comprises two components – LotusTrace, and LotusMap – that enable capturing preprocessing events and hardware analysis for preprocessing operations, respectively. The effectiveness of LotusTrace is due to the understanding of the PyTorch DataLoader's asynchronous data flow, which allows us to add logging instrumentation at the points that matter the most in capturing this flow (§ III-B [13]). As a result, LotusTrace neither performs additional computation nor maintains unnecessary tracer state in memory, thus avoiding CPU and memory overheads. On the other hand, LotusMap introduces a novel method that approximates the mapping of Python functions to their C/C++ counterparts. To obtain a high-quality mapping, our technique carefully buckets the C/C++ functions, filters incorrect C/C++ functions, and captures short-lived C/C++ functions (§ IV-B [13]).

Together, LotusTrace and LotusMap allow a practitioner to capture fine-grained batch-level preprocessing timing data, map them to the responsible C/C++ functions, and use their hardware performance counters to get a CPU architectural level performance view of the preprocessing operations. Lotus thus empowers users to reason about the performance of preprocessing pipelines at the hardware level, bridging a significant gap in our understanding.

3 Experiment Setup

We demonstrate Lotus' profiling capabilities using the Image Classification task as our primary example. Our full paper [13] provides additional studies using representative training tasks from the MLPerf training benchmark [35].

Image Classification (IC). This pipeline classifies an image to an object. We use MLPerf's reference PyTorch implementation [29, 35], the ImageNet dataset [18], and the ResNet18 [24] model. The pipeline contains the following preprocessing steps: 1. *Loader*: Loading the image from disk to memory and decoding it from compressed formats such as JPEG. 2. *RandomResizedCrop (RRC)*: Adjusting the image to the desired size and then crop. 3. *RandomHorizontalFlip (RHF)*: Obtaining mirror image. 4. *ToTensor (TT)*: Converting images to tensors. 5. *Normalization*: Normalizing to zero

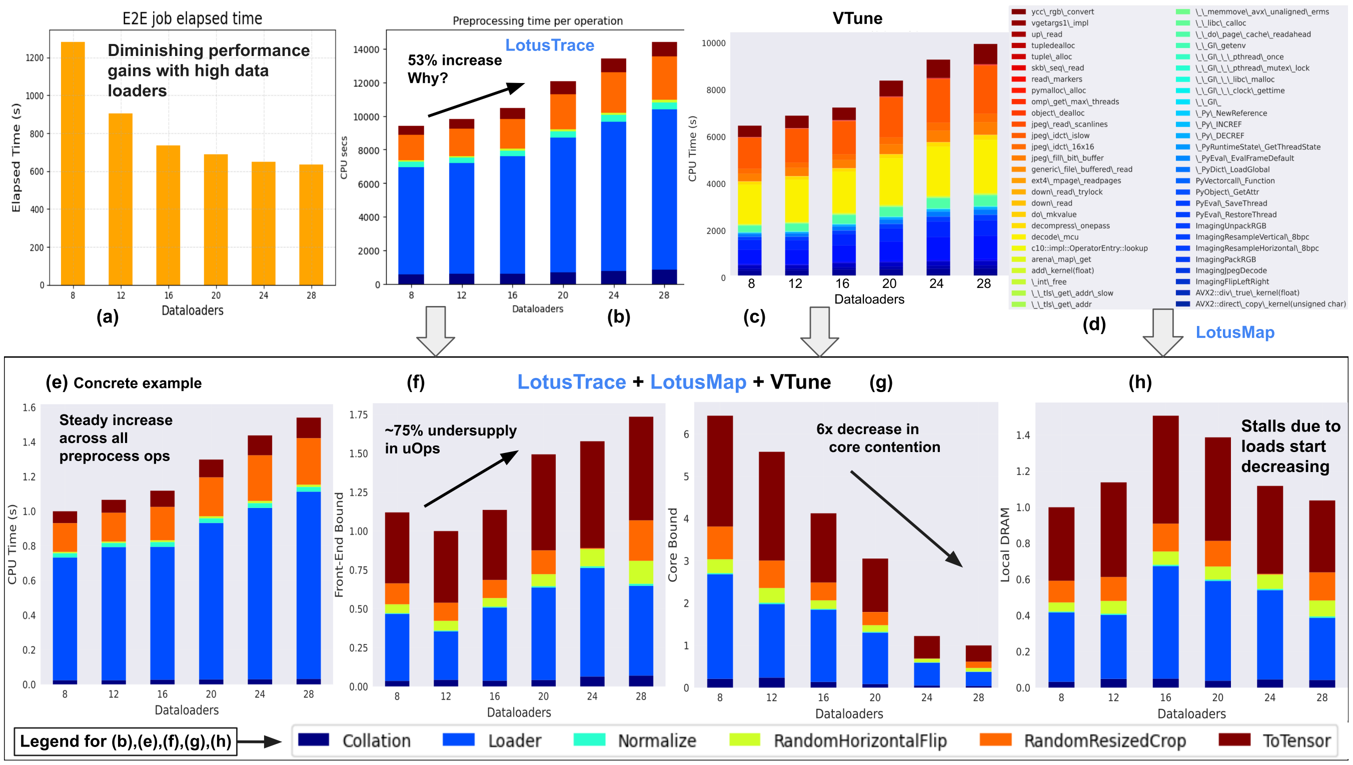

Figure 1. Combining LotusTrace and LotusMap enables analysis of performance of preprocessing operations on hardware.

mean and unit variance. 6. *Collation(C(k))*: Collating tensors into a batch size of k data elements.

Specifically, our experiment investigates the impact of the number of data loader workers on the image classification pipeline's performance. We use a fixed batch size of 1024 and 4 GPUs and vary the number of data loader workers from 8 to 28 in increments of 4. Exceeding 28 workers leads to OOM issues on our 32-core machine. The experiments run for 1 epoch, processing the same amount of training data across all configurations. As a result, the variability in preprocessing time is attributed to the number of dataloader workers. In our setup, preprocessing operations (including reading and decoding images) are CPU-based, whereas forward and backward passes on the deep learning model are GPU-based. Data collection involves using LotusTrace for preprocessing operation information and LotusMap with Intel VTune for hardware performance counter data.

Environment. The experiments are conducted on a CloudLab c4130 node [20], a dual-socket 3.2GHz E5-2667 Intel Xeon CPU, with 128 GiB of RAM, four NVIDIA V100 GPUs, each with 16 GiB memory and NVLink support, and a remote dataset mounted to a single node [17] as a ZFS zvol exported via iSCSI [1]. The software environment includes Python 3.10, PyTorch 2.0.1 with Torchvision 0.15, image processing using libjpeg-9e, GPU acceleration through CUDA 11.8 and cuDNN 8.7, and Ubuntu 20.04 (5.4.0-139-generic) for OS.

4 Observations from Hardware Performance

Figure 1 summarizes the profiling data obtained by Lotus. Overall, Lotus combines information collected via LotusTrace and LotusMap to link high-level Python functions with low-level hardware performance counters.

In Figure 1(a), we observe a ~50% drop in E2E job elapsed time as the number of dataloaders increase from 8 to 28. Beyond 20 dataloaders, there is a diminishing return in performance gain. LotusTrace reveals that total CPU seconds increased by 53% from 8 to 28 data loaders, with a steady rise in each preprocessing operation's CPU time (Figure 1(b)).

Since, VTune's profile collects hardware performance counters for C/C++ functions (over 300+) called during the run, it can not be directly used to explain the rise of CPU time for each preprocessing operation on the hardware level. We use LotusMap to obtain a mapping (Table I [13]) of C/C++ functions to Python preprocessing operations. This mapping allows us to filter out irrelevant C/C++ functions (Figure 1(c,d)). By combining the mapping and the elapsed time measured by LotusTrace, we can attribute hardware performance counters from C/C++ functions to the corresponding Python preprocessing operations, enabling reporting of hardware metrics per preprocessing operation (Figure 1(e - h)), *a capability not previously available.*

Figure 1(e) shows that CPU time increases steadily for all preprocessing operations, in line with our observation from LotusTrace. Figure 1(f) and Figure 1(g) further explain this increase by revealing a steep undersupply of uOperations to the backend as data loaders increase, causing low contention for cores in the backend of the microarchitecture. With the workload being front-end bound, the pressure on stalls caused by loads serviced by Local DRAM decreases (Figure 1(h)). This implies that in certain CPU SKUs, increasing the number of data loader workers does not translate to a decrease in E2E job elapsed time, as the CPU time increases due to the contention for hardware resources.

Additionally, this example underscores the importance of LotusMap's mapping quality. For instance, even though `ToTensor` is a short-lived function, it occurs frequently. Without capturing its mappings using techniques described in § IV-B [13], we wouldn't be able to account for its significant contribution to the trends observed in Figure 1(f,g,h).

***Takeaway**: Selecting a CPU SKU with the right number of cores is not straightforward, as adding cores may result in diminishing returns for reducing end-to-end job elapsed time, while increasing overall CPU time. Lotus helps identify hardware resource contention under various configurations.*

5 Conclusion

Lotus is an open-source profiling tool[1] for the preprocessing stage in ML pipelines which enables infrastructure designers to thoroughly evaluate hardware infrastructure under numerous configurations of the ML training job. Using Lotus requires small code changes, described in the system documentation, but these can be justified given the additional insights into the preprocessing pipeline execution. For instance, Lotus makes it possible to compare the efficiency of different CPU SKU choices for deploying the ML preprocessing stages, to identify CPU bottlenecks that could be addressed with new hardware designs, as well as to guide software-level optimizations. Future enhancements could integrate Lotus with existing accelerator profilers targeting the ML training execution, for an end-to-end view of the ML training pipeline.

Acknowledgement

This project is partially supported by the Intel Center on Transformative Server Architecture via the TRIM project, and the SRC/DARPA JUMP 2.0 Center for Processing with Intelligent Storage and Memories (PRISM).

References

[1] [n. d.]. Adding Zvols. https://www.truenas.com/docs/core/coretutorials/storage/pools/zvols/.
[2] [n. d.]. *Amazon Web Services (AWS) - Cloud Computing Services.* https://aws.amazon.com/
[3] [n. d.]. AMD μProf. https://www.amd.com/en/developer/uprof.html. Accessed: 2024-5-30.
[4] [n. d.]. *AWS Graviton Processors.* https://aws.amazon.com/ec2/graviton/
[5] [n. d.]. *Google Cloud Platform (GCP) - Cloud Computing Services.* https://cloud.google.com/
[6] [n. d.]. Intel VTune™ Profiler Documentation. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler-documentation.html. Accessed: 2024-5-30.
[7] [n. d.]. *Intel® Infrastructure Processing Unit (Intel® IPU).* https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html
[8] [n. d.]. *Microsoft Azure - Cloud Computing Services.* https://azure.microsoft.com/en-us/
[9] [n. d.]. NVIDIA Developer Data Loading Library (DALI). https://developer.nvidia.com/dali. Accessed: 2024-5-18.
[10] [n. d.]. TFRecord and tf.train.Example. https://www.tensorflow.org/tutorials/load_data/tfrecord. Accessed: 2024-5-18.
[11] [n. d.]. *torch.profiler.* https://pytorch.org/docs/2.0/profiler.html
[12] Andrew Audibert, Yang Chen, Dan Graur, Ana Klimovic, Jiří Šimša, and Chandramohan A Thekkath. [n. d.]. tf.data service: A Case for Disaggregating ML Input Data Processing. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (New York, NY, USA, 2023-10-30). ACM, 358–375.
[13] Rajveer Bachkaniwala, Harshith Lanka, Kexin Rong, and Ada Gavrilovska. 2024. Lotus: Characterization of Machine Learning Preprocessing Pipelines via Framework and Hardware Profiling. In *2024 IEEE International Symposium on Workload Characterization (IISWC).*
[14] Emery D Berger, Sam Stern, and Juan Altmayer Pizzorno. 2023. Triangulating Python Performance Issues with SCALENE. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23).* USENIX Association, Boston, MA, 51–64.
[15] BIZON. [n. d.]. *BIZON X9000 G2 – 8 GPU NVIDIA HGX A100, H100 SXM5 SXM4 Server with AMD EPYC, Intel Xeon – Ai, Deep Learning Server for Data Centers.* https://bizon-tech.com/bizon-x9000.html
[16] Dami Choi, Alexandre Passos, Christopher J Shallue, and George E Dahl. 2019. Faster Neural Network Training with Data Echoing. (July 2019). arXiv:1907.05550 [cs.LG]
[17] CloudLab. [n. d.]. CloudLab: 10 storage mechanisms. https://docs.cloudlab.us/advanced-storage.html.
[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition.* 248–255.
[19] Dihuni. [n. d.]. *GPU Server & Workstation Systems for AI – Dihuni – GPU Server for AI, Data Center & IoT Hardware & Software Solutions.* https://www.dihuni.com/product-category/gpu-systems/
[20] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC).* 1–14.
[21] Ben Frederickson. [n. d.]. py-spy: Sampling profiler for Python programs. https://github.com/benfred/py-spy.
[22] Dan Graur, Damien Aymon, Dan Kluser, Tanguy Albrici, Chandramohan A Thekkath, and Ana Klimovic. 2022. Cachew: Machine Learning Input Data Processing as a Service. In *2022 USENIX Annual Technical Conference (USENIX ATC 22).* USENIX Association, Carlsbad, CA, 689–706.
[23] Dan Graur, Oto Mraz, Muyu Li, Sepehr Pourghannad, Chandramohan A Thekkath, and Ana Klimovic. [n. d.]. Pecan: Cost-Efficient ML Data Preprocessing with Automatic Transformation Ordering and Hybrid Placement. In *2024 USENIX Annual Technical Conference (USENIX*

[1]Lotus is available at https://github.com/rajveerb/lotus

ATC 24) (Santa Clara, CA, 2024-07). USENIX Association, 649–665.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. (Dec. 2015). arXiv:1512.03385 [cs.CV]

[25] HPE. [n. d.]. *HPE Cray XD670.* https://www.hpe.com/us/en/hpe-cray-xd670.html

[26] HPE. [n. d.]. *The next era of supercomputing.* https://www.hpe.com/us/en/compute/hpc/supercomputing/cray-exascale-supercomputer.html

[27] Alexander Isenko, Ruben Mayer, Jeffrey Jedele, and Hans-Arno Jacobsen. 2022. Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1825–1839.

[28] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-Luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. [n. d.]. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2017-06-24). ACM.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[30] Michael Kuchnik, Ana Klimovic, Jiří Šimša, Virginia Smith, and George Amvrosiadis. 2022. Plumber: Diagnosing and Removing Performance Bottlenecks in Machine Learning Data Pipelines. In *Proceedings of Machine Learning and Systems*, D Marculescu, Y Chi, and C Wu (Eds.), Vol. 4. Systems and Machine Learning Foundation, Indio, CA, 33–51.

[31] G Leclerc, A Ilyas, L Engstrom, S Park, H Salman, and A Madry. 2023. FFCV: Accelerating Training by Removing Data Bottlenecks. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 12011–12020.

[32] Gyewon Lee, Irene Lee, Hyeonmin Ha, Kyunggeun Lee, Hwarim Hyun, Ahnjae Shin, and Byung-Gon Chun. 2021. Refurbish Your Training Data: Reusing Partially Augmented Samples for Faster Deep Neural Network Training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 537–550.

[33] Janusz Lisiecki and Michał Zientkiewicz. 2019. S9925: FAST AI DATA PREPROCESSING WITH NVIDIA DALI. https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9925-fast-ai-data-pre-processing-with-nvidia-dali.pdf. Accessed: 2024-5-15.

[34] Mark Lohmeyer and Parthasarathy Ranganathan. [n. d.]. *Why Google keeps building custom silicon: The story behind Axion.* https://cloud.google.com/transform/axion-arm-cpus-custom-silicon-advantage-history

[35] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debo Dutta, Udit Gupta, Kim Hazelwood, Andy Hock, Xinyuan Huang, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. [n. d.]. MLPerf Training Benchmark. In *Proceedings of Machine Learning and Systems* (2020), I Dhillon, D Papailiopoulos, and V Sze (Eds.), Vol. 2. 336–349.

[36] Microsoft. [n. d.]. *Announcing the preview of new Azure VMs based on the Azure Cobalt 100 processor.* https://techcommunity.microsoft.com/t5/azure-compute-blog/announcing-the-preview-of-new-azure-vms-based-on-the-azure/ba-p/4146353

[37] Microway. [n. d.]. *NVIDIA GPU Clusters.* https://www.microway.com/products/hpc-clusters/nvidia-gpu-clusters/

[38] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. 2021. Analyzing and Mitigating Data Stalls in DNN Training. *Proceedings VLDB Endowment* 14, 5 (Jan. 2021), 771–784.

[39] Derek G Murray, Jiří Šimša, Ana Klimovic, and Ihor Indyk. 2021. tf.data: A Machine Learning Data Processing Framework. *Proceedings VLDB Endowment* 14, 12 (July 2021), 2945–2958.

[40] NVIDIA. 2018. NVIDIA DGX-2 THE WORLD'S MOST POWERFUL DEEP LEARNING SYSTEM FOR THE MOST COMPLEX AI CHALLENGES. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-2/dgx-2-print-datasheet-738070-nvidia-a4-web-uk.pdf. Accessed: 2024-5-15.

[41] NVIDIA. 2019. NVIDIA DGX-1 THE ESSENTIAL INSTRUMENT FOR AI RESEARCH. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-1/dgx-1-rhel-datasheet-nvidia-us-808336-r3-web.pdf. Accessed: 2024-5-14.

[42] Pyeongsu Park, Heetaek Jeong, and Jangwoo Kim. [n. d.]. TrainBox: An extreme-scale neural network training server architecture by systematically balancing operations. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2020-10). IEEE, 825–838.

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. [n. d.]. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems* (2019), H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox, and R Garnett (Eds.), Vol. 32. Curran Associates, Inc.

[44] Ivan Svogor, Christian Eichenberger, Markus Spanring, Moritz Neun, and Michael Kopp. 2022. Profiling and Improving the PyTorch Dataloader for high-latency Storage: A Technical Report. *arXiv [cs.LG]* (Nov. 2022).

[45] Gabriele N Tornetta. [n. d.]. austin: A frame stack sampler for cpython. https://github.com/P403n1x87/austin.

[46] Przemek Tredak and Simon Layton. [n. d.]. S8906: Fast Data Pipelines for Deep Learning Training, 2018. https://www.nvidia.com/en-us/on-demand/session/gtcsiliconvalley2018-s8906/.

[47] Taegeon Um, Byungsoo Oh, Byeongchan Seo, Minhyeok Kweun, Goeun Kim, and Woo-Yeon Lee. 2023. Fastflow: Accelerating deep learning model training with smart offloading of input data pipeline. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1086–1099.

[48] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Mingxia Li, Fan Yang, Qianxi Zhang, Binyang Li, Yuqing Yang, Lili Qiu, Lintao Zhang, and Lidong Zhou. 2023. SiloD: A Co-design of Caching and Scheduling for Deep Learning Clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 883–898.

[49] Hanyu Zhao, Zhi Yang, Yu Cheng, Chao Tian, Shiru Ren, Wencong Xiao, Man Yuan, Langshi Chen, Kaibo Liu, Yang Zhang, Yong Li, and Wei Lin. 2023. GoldMiner: Elastic Scaling of Training Data Pre-Processing Pipelines for Deep Learning. *Proc. ACM SIGMOD Int. Conf. Manag. Data* 1, 2 (June 2023), 1–25.

[50] Mark Zhao, Emanuel Adamiak, and Christos Kozyrakis. 2024. cedar: Composable and Optimized Machine Learning Input Data Pipelines. (Jan. 2024). arXiv:2401.08895 [cs.LG]

[51] Mark Zhao, Niket Agarwal, Aarti Basant, Buğra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 1042–1057.