A Selective Preprocessing Offloading Framework for Reducing Data Traffic in DL Training

Meng Wang
University of Chicago
Chicago, IL, USA
wangm12@uchicago.edu

Gus Waldspurger
University of Chicago
Chicago, IL, USA
gusw@uchicago.edu

Swaminathan Sundararaman
IBM Research
San Jose, CA, USA
swami@ibm.com

Abstract

Deep learning (DL) training is data-intensive and often bottlenecked by fetching data from remote storage. Recognizing that many samples' sizes diminish during data preprocessing, we explore selectively offloading preprocessing to remote storage to mitigate data traffic. We conduct a case study to uncover the potential benefits and challenges of this approach. We then propose SOPHON, a framework that selectively offloads preprocessing tasks at a fine granularity in order to reduce data traffic, utilizing online profiling and adaptive algorithms to optimize for every sample in every training scenario. Our results show that SOPHON can reduce data traffic and training time by 1.2-2.2x over existing solutions.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Cloud computing**.

Keywords: Systems for DL Training, Cloud Storage, Preprocessing Offloading

ACM Reference Format:
Meng Wang, Gus Waldspurger, and Swaminathan Sundararaman. 2024. A Selective Preprocessing Offloading Framework for Reducing Data Traffic in DL Training. In *16th ACM Workshop on Hot Topics in Storage and File Systems (HOTSTORAGE '24), July 8–9, 2024, Santa Clara, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3655038.3665947

1 Introduction

Deep Learning (DL) has become a pivotal technology across various domains, including computer vision, natural language processing, and audio processing [1–5]. Concurrently, cloud platforms have increasingly come to provide specialized services to facilitate DL training [6–8].

DL training is characterized by its intense computational and data requirements [9–21], necessitating substantial storage for vast datasets, CPU resources for data preprocessing, and powerful GPUs for executing complex neural network models. The sheer volume of data used in DL training, now scaling to tens or even hundreds of terabytes [20, 22], surpasses the local storage capacities of most cloud compute nodes. As a result, cloud DL training typically decouples storage from compute, executing training on compute nodes that fetch data from remote storage clusters such as distributed file systems or object stores [23–26] to handle the extensive data volume requirements.

However, the bandwidth for data transfer from remote storage to compute nodes is often constrained, frequently becoming a critical bottleneck for DL workloads and limiting GPU utilization [18–21]. As GPUs become faster, this data fetch bottleneck becomes increasingly problematic.

A variety of strategies have been proposed to mitigate data fetch bottlenecks. Existing approaches mainly focus on selectively caching data in local storage or memory [16–20, 27, 28]. However, these methods are limited by the capacities of local storage and memory—a limitation that becomes more pronounced as datasets continue to increase in size. Other solutions explore storing preprocessed dataset in remote storage for repeated use across all epochs [29–31]. However, this approach risks compromising training accuracy, since the variability introduced by random data transformations during preprocessing is crucial for effective learning.

While previous studies [32–36] have delved into offloading preprocessing, their focus has been on alleviating CPU-based preprocessing bottlenecks by offloading tasks to additional CPU nodes, missing the opportunity to mitigate data fetch traffic. Furthermore, many of these works offload preprocessing operations en masse, and all of them apply a uniform approach across all data samples. This coarse-grained approach overlooks the fact that, as we demonstrate in our paper, *a great portion* of samples reach a minimum size during *intermediate* stages of the preprocessing pipeline.

In our paper, we utilize preprocessing offloading differently: *by observing that many data samples exhibit size reductions at intermediate stages of the preprocessing pipeline, we propose to* **selectively offload** *parts of preprocessing for certain samples to the remote storage server.* Given that the bandwidth within a storage cluster (intra-cluster) is generally

HOTSTORAGE '24, July 8–9, 2024, Santa Clara, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0630-1/24/07.
https://doi.org/10.1145/3655038.3665947

	Operation Selective	Data Partial	Data Selective	To Near Storage
tf.data svc	–	–	–	–
GoldMiner	✔	–	–	–
FastFlow	–	✔	–	–
cedar	✔	–	–	–
SOPHON	✔	✔	✔	✔

Table 1. Existing Offloading [32–35] vs. SOPHON.

much greater than that between the storage and compute clusters (inter-cluster) [37–40], our approach seeks to preprocess larger samples within the storage cluster itself. This methodology aims to minimize data traffic between remote storage and compute nodes by transmitting smaller, partially preprocessed samples, thereby optimizing network usage and enhancing training efficiency.

Crafting the optimal strategy for preprocessing offloading is complex. Our analysis of specific DL workloads' prepreprocessing pipelines reveals several insights: not every workload necessitates offloading, not every sample benefits from it, and offloading can incur varying levels of additional CPU load on the remote server, potentially creating new bottlenecks. Thus, an effective solution must address three critical questions: (1) Does a particular workload require preprocessing offloading to mitigate network traffic? (2) Which specific samples should undergo offloading? (3) For those samples identified, which preprocessing operations are most suitable for offloading?

In response, we introduce SOPHON (Selectively Offloading Preprocessing with Hybrid Operations Near-storage), a framework designed to selectively offload DL preprocessing tasks to remote storage servers with the goal of reducing data transfer traffic. SOPHON has two key components: (1) A two-stage profiler that collects essential metrics for making offloading decisions. Offloading is activated only when the workload is identified as I/O-bound during profiling. (2) A decision engine that determines which samples to offload and identifies the specific operations to offload for each chosen sample, striking a balance between reducing traffic and managing CPU overhead. Together, these components enable SOPHON to provide tailored offloading strategies to meet the unique needs and constraints of each training scenario.

As **Table 1** shows, SOPHON is the first work that implements data-selective offloading for DL preprocessing, where "data-selective" refers to selecting specific samples for offloading based on each individual sample's characteristics.

Our evaluation results demonstrate that SOPHON can effectively enhance training efficiency, achieving a 1.2-2.2x reduction in training time over existing solutions.

SOPHON is tailored for specific DL training workloads where remote IO is a bottleneck. It may not offer advantages in certain scenarios, such as with Large Language Models. Further details are discussed in Section 5.

We also discuss future work in Section 6.

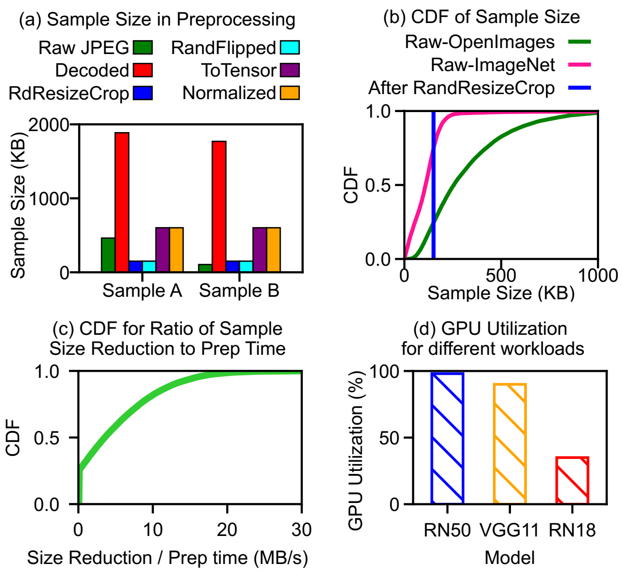

Figure 1. Analysis of Preprocessing Pipeline. *The figures are explained in Section 2.*

2 Preprocessing Analysis

A typical DL training iteration entails: (1) Fetching data from storage, (2) Preprocessing data on CPUs, (3) Transferring data to GPUs, (4) Forward propagation for predictions, (5) Backward propagation for parameter updates.

In this section, we analyze the preprocessing pipeline of a specific DL workload, revealing opportunities and challenges in minimizing data traffic through strategic offloading of preprocessing tasks. Our investigation is anchored in a case study on image classification workloads. We employ the official PyTorch example training script, sourced from its GitHub repository [41], conducting experiments on subsets of two key datasets in computer vision DL research: OpenImages [22] and ImageNet [42].

The workload's preprocessing pipeline consists of five key operations: (1) **Decode**: Converts raw binary (e.g., JPEG) to an image object for manipulation via Python libraries like PIL; (2) **RandomResizedCrop**: Crops a random image section and resizes it to a specified size; (3) **RandomHorizontalFlip**: Flips the image horizontally at random; (4) **ToTensor**: Transforms the PIL Image from a uint8 ([0,255]) list to a float Tensor, scaling to [0.0, 1.0]. (5) **Normalize**: Normalizes the tensor image with mean and standard deviation.

By measuring file sizes before and after each preprocessing step and assessing the preprocessing time cost, we arrived at several key findings:

Finding #1: Variations in file size across the preprocessing steps highlight opportunities for reducing data transfers through preprocessing offloading. For example, as illustrated in **Figure 1a**, Sample A's size drops from 462KB raw JPEG to 151KB post RandomResizedCrop—when the image is cropped and resized to 224x224 pixels, with each pixel's R/G/B value represented by 1 byte. This suggests that offloading Decode and RandomResizedCrop to the remote storage server before network transmission can notably reduce data traffic.

Finding #2: The observation that the minimum file size occurs before the final preprocessing step suggests a decomposed preprocessing offloading approach. Existing offloading frameworks often consider the entire preprocessing as a singular unit [32, 33], missing opportunities for data traffic reduction. As depicted in **Figure 1a**, the file size post-Normalize is 4x larger than after RandomResizedCrop and RandomHorizontalFlip, due to the ToTensor operation converting pixel values from 1 byte per R/G/B to 4 byte floats. Thus, to minimize network traffic, an offloading framework should permit the selective offloading of specific preprocessing operations.

Finding #3: The varied impact of preprocessing offloading across different images necessitates finer-grained decision making. For instance, Sample B's smallest file size occurs in its raw JPEG format, as shown in **Figure 1a**. Thus, unlike Sample A, Sample B would transfer more efficiently without any preprocessing offloading. As further illustrated by **Figure 1b**, while 76% of OpenImages exhibit size reductions post certain preprocessing operations, 24% are smallest in their raw JPEG format and should not undergo offloading. A similar pattern is observed with ImageNet, with 26% of images benefiting from offloading, while 74% do not, underscoring the need for a tailored offloading strategy.

Finding #4: Preprocessing traffic reductions have varying CPU costs. Reflecting on previous research, preprocessing tasks are notably CPU-intensive [17, 32, 33]. When such tasks are offloaded to remote storage nodes, which typically possess lesser CPU capabilities compared to compute nodes, considerable CPU overhead is incurred. This scenario highlights the need for weighing this overhead against efficiency gains. To better understand this tradeoff, we measured preprocessing time for each operation and image in the 12GB subset, with the raw data cached in memory and processed using 8 CPU cores running in parallel. For each image, we then calculated the preprocessing time cost needed to offload in order to reach the minimum data transfer.

As illustrated in **Figure 1c**, the ratio of file size reduction to preprocessing time across the OpenImages dataset serves as an indicator of the trade-off between network traffic savings and the CPU time cost. Notably, 24% of images attain their minimum size in raw JPEG and thus exhibit a ratio of 0 and do not need offloading. For the remaining 76% of images, this ratio varies, necessitating a refined offloading strategy that prioritizes images yielding the highest network traffic savings per unit of CPU time, particularly when CPU resources at the storage node are limited.

Finding #5: DL workloads exhibit varying demands for preprocessing offloading, which calls for customized offloading decisions. As depicted in **Figure 1d**, the GPU utilization across three distinct models—trained using the same configuration of a V100 GPU, ample CPUs, and constrained bandwidth to remote storage—is different. Specifically, ResNet50, with high GPU compute intensity, achieves near-maximal GPU utilization, rendering it less susceptible to gains from preprocessing offloading. Conversely, ResNet18, with lighter GPU compute requirements, spends about 65% of its time in a data-fetching idle state, suggesting considerable offloading benefits. Hence, the decision regarding preprocessing offloading should not only factor in image size and preprocessing time but also account for the resource demands and characteristics of each workload.

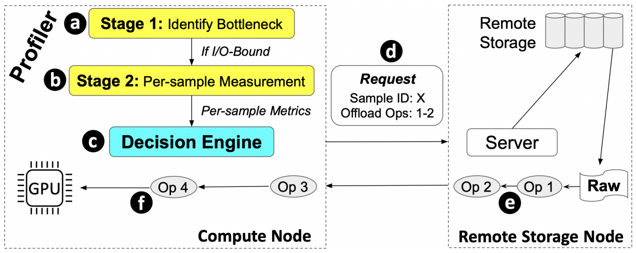

Figure 2. SOPHON Design Overview.

3 Design

Based on Finding #1, there exists a substantial opportunity to improve data fetching efficiency in DL training through the strategic offloading of preprocessing tasks. However, Findings #2-5 also highlight several challenges that need to be addressed in order to fully capitalize on these opportunities. An efficient offloading framework must: (1) assess the need for preprocessing offloading to mitigate network traffic for specific workloads, (2) choose appropriate data samples for preprocessing offloading, and (3) select the precise preprocessing operations to offload for each sample.

To tackle these challenges, we introduce SOPHON (Selectively Offloading Preprocessing with Hybrid Operations Near-storage), a solution engineered to selectively offload DL preprocessing to remote storage servers, aiming at minimizing data transfer traffic. SOPHON is designed to systematically navigate each of these decision points, utilizing online data analysis and adaptive algorithms to tailor offloading decisions to the unique demands of each training scenario.

SOPHON has two key components: a two-stage profiler that collects essential metrics for informed offloading decisions, and a decision engine that determines the optimal offloading strategy using these metrics.

Figure 2 illustrates how SOPHON works. ⓐ The profiler first assesses GPU, CPU, and I/O throughput to determine if the given DL workload needs offloading to reduce I/O constraints. ⓑ If the workload is I/O-bound, SOPHON moves to the second profiling stage, noting the time and size changes for each preprocessing step across all samples. ⓒ With detailed metrics and knowledge of compute and storage node resources, SOPHON formulates the best offloading plan per sample. ⓓ Offloading directives for each sample are then incorporated into data fetch requests to the storage server, detailing the specific operations for offloading. ⓔ The storage server processes these operations as instructed, sending back the partially processed data to the compute node. (f) Finally, the compute node finishes any remaining preprocessing and forwards the data to GPUs for training.

3.1 Two-Stage Profiler

To minimize the profiling overhead while collecting essential metrics for informed offloading decisions, our approach harnesses a two-stage profiling process.

Inspired by Finding #5 and borrowing the idea from [17], the first stage briefly assesses the primary bottleneck within the workload by measuring GPU, I/O, and CPU throughput. This is achieved by executing 50 batches under three distinct settings: (1) model training on the GPU using synthetic data to eliminate CPU or I/O delays, (2) data retrieval from remote storage, devoid of CPU or GPU processing to isolate I/O throughput, and (3) CPU-intensive preprocessing on the data cached during the second setting to gauge CPU throughput. This approach provides insights into the workload's throughput demands with minimal overhead (a typical training job spans over 50 epochs, each with thousands of batches). If the workload is I/O-bound, SOPHON proceeds to the second profiling stage; otherwise, it defaults to the standard training without offloading. CPU-bound scenarios may benefit from other solutions to mitigate preprocessing delays [32, 33].

In the second stage, we collect details on CPU time for preprocessing and changes in each sample's size through all preprocessing operations. This stage, requiring detailed, sample-specific data, involves processing the entire dataset, potentially incurring significant overhead. To minimize this, we use an on-the-fly profiling method: we proceed with the first training epoch without offloading any preprocessing tasks and collect essential per-sample metrics. This approach effectively lowers profiling overhead, facilitating efficient acquisition of performance data.

We currently assume identical CPU types on compute and storage nodes, allowing preprocessing times profiled on the compute node to be used for the storage node. We plan to explore heterogeneous CPU scenarios in the future.

3.2 Offloading Policy

Leveraging the metrics obtained through comprehensive profiling, SOPHON determines the most advantageous offloading strategy based on Findings #2-4.

Our strategy evaluates the potential size reduction and required preprocessing time for each data sample to reach its minimum size. Samples showing size reduction compared to their raw forms are considered for offloading. SOPHON measures *offloading efficiency* by the ratio of size reduction to preprocessing time, where a higher ratio suggests better potential for data traffic reduction per CPU time spent. Therefore, samples are selected for offloading in descending order of efficiency, prioritizing those with the most significant effect on reducing data traffic.

SOPHON makes the decision based on four key metrics: (1) $\mathbf{T_G}$: The GPU time for one training epoch; (2) $\mathbf{T_{CC}}$: The CPU time on the compute node for local preprocessing, calculated as the total local preprocessing time divided by the CPU core count; (3) $\mathbf{T_{CS}}$: The CPU time on the storage node for offloaded preprocessing tasks, determined by dividing the total offloaded preprocessing time by the storage node's available CPU core count; (4) $\mathbf{T_{Net}}$: The time for data transfer from remote storage to the compute node over one epoch, derived from the total data traffic and the network bandwidth.

The first step uses a baseline profile without any offloading, characterized by T_{Net} as the predominant metric due to the I/O-bound nature of the workload and T_{CS} being 0 (no offloading). From this point, SOPHON selects the sample with the highest offloading efficiency, effecting a reduction in both T_{CC} and T_{Net} while elevating T_{CS}. The goal is to aggressively minimize network traffic until it ceases to be the limiting factor. This iterative selection of high-efficiency samples continues until either of two conditions is met: (1) T_{Net} ceases to be the predominant metric, or (2) no further samples with positive offloading efficiency remain. Through this algorithm, SOPHON minimizes network traffic without imposing excessive preprocessing load on the storage server.

3.3 Why Not Preprocess Just Once

One could contemplate a strategy where samples are selectively preprocessed just once to minimum size and then stored for reuse across all epochs. While this simplifies the process, it risks diminishing training accuracy. Random augmentations, typically applied during online preprocessing, are crucial for DL training accuracy and should be performed in each epoch. In contrast, our solution retains the original training's preprocessing logic and preserves accuracy.

4 Evaluation

We implement SOPHON in Python on top of PyTorch 1.8.0 in around 990 LOC. We utilize the gRPC framework to facilitate communication for data fetch requests and responses.

We evaluate SOPHON with small-scale experiments to quickly gauge its performance and demonstrate its benefits. We plan to use more realistic settings in the future.

Workload Benchmarks: We benchmark the performance of SOPHON on image classification tasks within DL training. We utilize the official PyTorch example training script, taken directly from its GitHub repository [41]. We train the AlexNet model, which is characterized as compute-light and thus is easily bottlenecked by data fetching. We conduct experiments on subsets of two popular datasets in computer vision DL research: a 12GB subset of OpenImages [22] and an 11GB subset of ImageNet [42].

Experiment Setup: We use a two-node setup: one as the compute node equipped with an RTX-6000 GPU , and the other as the storage server node. Both nodes utilize two Intel Xeon Gold 6126 @ 2.60GHz processors. On the compute node, we allocate 48 logical cores to eliminate preprocessing bottlenecks in the original workload, thereby making I/O the bottleneck. The storage node's CPU allocation is varied to assess the impact of CPU overhead introduced by offloading preprocessing. To simulate a bandwidth-constrained environment, we cap the network throughput at 500 Mbps. Furthermore, on the storage node, datasets are cached in memory to mirror the frequently assumed condition where intra-cluster bandwidth vastly exceeds inter-cluster bandwidth [37–40].

Simulation of Real-World Configuration: While we use small subsets that technically could fit into local storage, our experiments consistently fetch data from the remote storage node. This mimics real-world scenarios where datasets exceed local storage capacities. For example, the full OpenImages dataset totals 18TB [19, 43]. Each subset used in our experiments comprises over 40,000 images, randomly selected from the original dataset to represent the variety in sizes and preprocessing costs found in the full dataset.

We limit the network bandwidth to 500 Mbps to introduce a remote I/O bottleneck even when training with a single RTX-6000 GPU. In practical scenarios, training ResNet50 on ImageNet with 8 V100 GPUs requires nearly 16 Gbps of I/O bandwidth to fully utilize GPUs [19, 20], and even a 10 Gbps network could cause a significant remote I/O bottleneck.

Additionally, on the storage node, we cache the datasets in memory to ensure that the intra-node data read bandwidth is much higher than the network bandwidth between our two nodes. This setup mimics typical real-world conditions where the aggregate intra-cluster bandwidth of distributed storage systems greatly surpasses the inter-cluster network bandwidth, a common assumption in prior research [37–40].

Baselines: We establish several baselines for comparison: **No-Off**, the original training pipeline without preprocessing offloading; **All-Off**, with all preprocessing operations of all samples offloaded to the storage node; **FastFlow** [33], a preprocessing offloading framework designed to alleviate CPU bottlenecks, which treats all preprocessing operations as a single unit and does not differentiate between data samples; and **Resize-Off**, which offloads only the Decode and RandomResizedCrop operations to the storage node, based on the observation that resizing reduces many images' sizes.

Our evaluation of SOPHON spans two distinct scenarios: one with ample CPU cores at the remote storage, and another where CPU resources are limited.

4.1 Ample CPU Cores on Storage Node

We start our evaluation using a storage node with ample (48) CPU cores to maximize the benefits of preprocessing offloading. **Figure 3** displays both training times and data traffic per epoch for all offloading policies.

All-Off has the longest training time across all policies, increasing data traffic by 1.9x for OpenImages and 5.1x for ImageNet when compared to No-Off. This is due to the data being converted into float tensors, which results in substantially larger sizes for most samples.

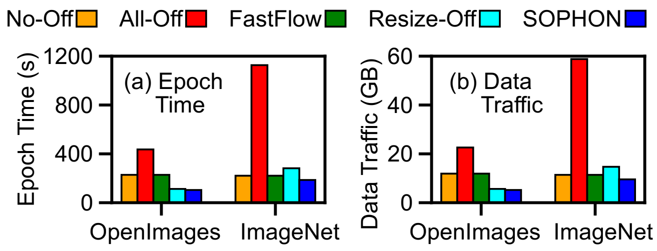

Figure 3. SOPHON's effect with ample CPU cores. *SOPHON cuts traffic/epoch time when storage node has ample CPUs.*

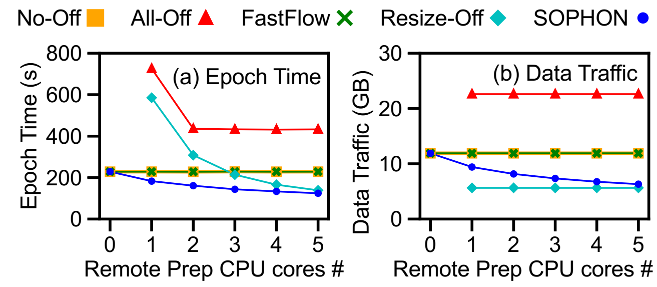

Figure 4. SOPHON's effect with limited CPU cores. *SOPHON finds the best solution when storage node has limited CPUs.*

FastFlow chooses to not offload preprocessing in the evaluated setups. This decision is informed by its coarse-grained profiling analysis, which indicates that offloading all operations would lead to increased training time.

Resize-Off reduces data traffic by 2x for the OpenImages dataset compared to No-Off due to the large raw sizes of most images (76% of samples become smaller after Decode and RandomResizedCrop.) *However, Resize-Off increases traffic by 1.3x compared to No-Off for the ImageNet dataset* due to its smaller average image size (only 26% of samples become smaller after Decode and RandomResizedCrop.)

SOPHON improves performance for both datasets, thanks to its fine-grained offloading. For OpenImages, SOPHON achieves a 2.2x reduction in data traffic compared to No-Off, outperforming Resize-Off by not offloading preprocessing for samples that do not benefit. Unlike Resize-Off, SOPHON manages to reduce data traffic by 1.2x for ImageNet, thereby reducing training time. Selectively offloading preprocessing steps based on which images become smaller during preprocessing allows SOPHON to treat each sample optimally.

4.2 Limited CPU Cores on Storage Node

Next, we evaluate SOPHON's efficacy by varying the number of CPU cores allocated for preprocessing on the storage node. We focus on the OpenImages dataset, which has demonstrated greater benefits from preprocessing offloading. We don't include results for the ImageNet dataset due to space constraints. **Figure 4** shows the results for OpenImages.

All-Off results in the longest training time due to increased data traffic. Additionally, training time is further increased when only 1 CPU core is allocated for remote preprocessing, as the remote CPU overhead creates a bottleneck.

FastFlow consistently decides against preprocessing offloading as it anticipates that offloading *all* operations would increase the training time.

Resize-Off achieves the lowest data traffic among all configurations. However, its training time is not optimal as it offloads an excessive amount of preprocessing to the remote server, causing CPU overhead to create a new bottleneck. When the storage node has ≤ 2 CPU cores available for preprocessing, Resize-Off performs even worse than No-Off.

Finally, SOPHON exhibits the shortest training time among all policies, effectively balancing the trade-off between data traffic reduction and offloaded CPU overhead. Notably, there are diminishing returns for training time when allocating additional CPU cores. For instance, the transition from 0 cores to 1 leads to a 22-second reduction in epoch time, whereas moving from 4 to 5 CPU cores results in only a 9-second reduction. This demonstrates SOPHON's proficiency in selecting samples for offloading with the highest efficiencies, optimizing outcomes even under CPU constraints.

5 Discussion

While our work manages to reduce remote data traffic for specific DL training workloads, we understand that it might not help in some scenarios. Below we discuss the use cases of SOPHON and scenarios where it might not be beneficial.

Remote I/O bottleneck in DL Training: DL training often involves large datasets to enhance model accuracy. For instance, the Google OpenImages dataset totals 18TB [19, 43]. Such large datasets often exceed local storage capacities, leading to the use of remote cloud storage services to fetch data during training. In some GPU clusters, 97.3% of DL jobs store their data in cloud storage [20], resulting in a separation between GPU clusters and remote storage.

The rapid advancement in GPU computation speeds necessitates high-speed data transfers to avoid data stalls [16–20, 27, 28]. Furthermore, GPU clusters often run hundreds or thousands of DL training jobs simultaneously, putting substantial strain on the network between GPU clusters and remote storage. For example, a 400 V100 GPU cluster requires an aggregate I/O bandwidth of 200Gbps [20], while Azure's maximum egress bandwidth is only 120Gbps [44]. This remote I/O bottleneck is likely to worsen in the future due to the fast evolvement of GPUs [20].

SOPHON is effective for such scenarios where remote I/O bottlenecks may occur, as it reduces remote data traffic.

DL Training with Essential Need for Online Preprocessing: Many DL training jobs, especially computer vision models, require online preprocessing to enhance training accuracy. These workloads provide opportunities for SOPHON to reduce data traffic via selective preprocessing offloading.

Near Storage Processing Support: Modern cloud storage services increasingly support near-storage data processing, facilitating the offloading strategies of SOPHON. For instance, Ceph enables near-storage data processing through dynamic object interfaces [45]. Similarly, Amazon S3 Object Lambda allows users to submit custom data processing code that is executed automatically before data is returned [46].

Scenarios Where SOPHON Might Not Work: SOPHON may not help for Large Language Models (LLMs), where input data preprocessing is less critical for accuracy, limiting opportunities for preprocessing offloading. Though LLMs are becoming more popular, a substantial number of DL training jobs still exist in current clusters. This is because LLMs are often very expensive to run, while many DL models can achieve satisfactory accuracy at a much lower cost.

SOPHON assumes CPU-based preprocessing and currently doesn't support GPU-based strategies like NVIDIA DALI [47]. However, we believe our findings also points to new opportunities in GPU-based preprocessing scenarios. For example, one can selectively split preprocessing tasks between GPUs and CPUs to reduce CPU-GPU data transfers.

SOPHON doesn't help when the entire dataset can fit into local storage and thus remote I/O is not needed. For such training scenarios, many prior works have focused on alleviating the potential local I/O bottleneck through effcient caching and prefetching strategies[16, 17, 27].

6 Conclusion and Future Work

We reveal opportunities and challenges in reducing data traffic through strategic preprocessing offloading in DL training. We propose SOPHON, a framework that selectively offloads preprocessing tasks to minimize data traffic, utilizing online profiling and adaptive algorithms to optimize for every sample. Our preliminary results demonstrate that SOPHON can effectively reduce data traffic and training time.

Future Work: We plan to design a strategy to selectively compress preprocessed data, further reducing data traffic while considering potential CPU overhead increases. Additionally, we aim to extend support to environments with heterogeneous CPU types across compute and storage nodes.

Furthermore, we intend to explore the implementation of SOPHON in multi-tenant environments. We plan to develop a scheduler to efficiently allocate storage-side CPUs among multiple jobs, maximizing global training efficiency.

We will also study a wider variety of DL training workloads across various domains and conduct evaluations in more comprehensive settings.

7 Acknowledgments

We thank Jayashree Mohan, our shepherd, and the anonymous reviewers for their tremendous feedback and comments. This work was supported by funding from NSF (grant Nos. CCF-2119184 and CNS-2027170).

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 26th Conference on Neural Information Processing Systems (NIPS)*, 2012.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[4] Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Google Cloud Deep Learning VM Images. https://cloud.google.com/deep-learning-vm.

[7] Azure Machine Learning - ML as a Service. https://azure.microsoft.com/en-us/products/machine-learning.

[8] Amazon SageMaker. https://aws.amazon.com/sagemaker/.

[9] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. Multi-resource interleaving for deep learning training. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2022.

[10] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proceedings of the 2018 EuroSys Conference (EuroSys)*, 2018.

[11] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters. In *Proceedings of the 16th Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.

[12] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. In *Proceedings of the 14th Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[13] Lijie Xu, Shuang Qiu, Binhang Yuan, Jiawei Jiang, Cedric Renggli, Shaoduo Gan, Kaan Kara, Guoliang Li, Ji Liu, Wentao Wu, Jieping Ye, and Ce Zhan. In-Database Machine Learning with CorgiPile: Stochastic Gradient Descent without Full Data Shuffle. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2022.

[14] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, , and Lidong Zhou. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *Proceedings of the 13th Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[15] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, , and S. Viswanatha. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *Proceedings of the 2020 EuroSys Conference (EuroSys)*, 2020.

[16] Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. FFCV: Accelerating Training by Removing Data Bottlenecks. In *2023 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[17] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and Mitigating Data Stalls in DNN Training. In *Proceedings of the 47th International Conference on Very Large Databases (VLDB)*, 2021.

[18] Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. Clairvoyant prefetching for distributed machine learning I/O. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

[19] Abhishek Vijaya Kumar and Muthian Sivathanu. Quiver: An Informed Storage Cache for Deep Learning. In *Proceedings of the 18th USENIX Symposium on File and Storage Technologies (FAST)*, 2020.

[20] 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. SiloD: A Co-design of Caching and Scheduling for Deep Learning Clusters. In *Proceedings of the 2023 EuroSys Conference (EuroSys)*, 2023.

[21] Lipeng Wang, Songgao Ye, Baichen Yang, Youyou Lu, Hequan Zhang, Shengen Yan, and Qiong Luo. DIESEL: A Dataset-Based Distributed Storage and Caching System for Large-Scale Deep Learning Training. In *49st International Conference on Parallel Processing (ICPP)*, 2020.

[22] Open Images Dataset V7 and Extensions. https://storage.googleapis.com/openimages/web/index.html.

[23] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST)*, 2010.

[24] GlusterFS. https://www.gluster.org.

[25] Azure Blob Storage. https://azure.microsoft.com/en-us/products/storage/blobs.

[26] Amazon S3. https://aws.amazon.com/s3/.

[27] Derek G. Murray, Jiří Šimša, Ana Klimovic, and Ihor Indyk. tf.data: A Machine Learning Data Processing Framework. In *Proceedings of the 47th International Conference on Very Large Databases (VLDB)*, 2021.

[28] Chih-Chieh Yang and Guojing Cong. Accelerating Data Loading in Deep Neural Network Training. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2019.

[29] Alexander Isenko, Ruben Mayer, Jeffrey Jedele, and Hans-Arno Jacobsen. Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2022.

[30] Aarati Kakaraparthy, Abhay Venkatesh, Amar Phanishayee, and Shivaram Venkataraman. The Case for Unifying Data Loading in Machine Learning Clusters. In *The 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2019.

[31] Gyewon Lee, Irene Lee, Hyeonmin Ha, Kyung-Geun Lee, Hwarim Hyun, Ahnjae Shin, and Byung-Gon Chun. Refurbish Your Training Data: Reusing Partially Augmented Samples for Faster Deep Neural Network Training. In *Proceedings of the 2021 USENIX Annual Technical Conference (ATC)*, 2021.

[32] Andrew Audibert, Yang Chen, Dan Graur, Ana Klimovic, Jiří Šimša, and Chandramohan A. Thekkath. tf.data service: A Case for Disaggregating ML Input Data Processing. In *Proceedings of the 14th ACM Symposium on Cloud Computing (SoCC)*, 2023.

[33] Taegeon Um, Byungsoo Oh, Byeongchan Seo, Minhyeok Kweun, Goeun Kim, and Woo-Yeon Lee. FastFlow: Accelerating Deep Learning Model Training with Smart Offloading of Input Data Pipeline. In *Proceedings of the 49th International Conference on Very Large Databases (VLDB)*, 2023.

[34] 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. GoldMiner: Elastic Scaling of Training Data Pre-Processing Pipelines for Deep Learning. In *Proceedings of the 2023 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2023.

[35] Mark Zhao, Emanuel Adamiak, and Christos Kozyrakis. cedar: Composable and optimized machine learning input data pipelines. *arXiv preprint arXiv:2401.08895*, 2024.

[36] 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 Po. Understanding Data Storage and Ingestion for Large-Scale Deep Recommendation Model Training. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022.

[37] Shushi Gu, Fugang Wang, Qinyu Zhang, Tao Huang, and Wei Xiang. Global repair bandwidth cost optimization of generalized regenerating codes in clustered distributed storage systems. *IET Communications*, 15(19):2469–2481, 2021.

[38] Vitaly Abdrashitov, N. Prakash, and Muriel Médard. The storage vs repair bandwidth trade-off for multiple failures in clustered storage networks. In *IEEE Information Theory Workshop (ITW)*, 2017.

[39] Yuchong Hu, Liangfeng Cheng, Qiaori Yao, Patrick P. C. Lee, Weichun Wang, and Wei Chen. Exploiting Combined Locality for Wide-Stripe Erasure Coding in Distributed Storage. In *Proceedings of the 19th USENIX Symposium on File and Storage Technologies (FAST)*, 2021.

[40] Meng Wang, Jiajun Mao, Rajdeep Rana, John Bent, Serkay Olmez, Anjus George, Garrett Wilson Ransom, Jun Li, , and Haryadi S. Gunawi. Design Considerations and Analysis of Multi-Level Erasure Coding in Large-Scale Data Centers. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2023.

[41] PyTorch ImageNet example training script. ImageNet training in PyTorch. https://github.com/pytorch/examples/tree/main/imagenet.

[42] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. In *International Journal of Computer Vision (IJCV)*, 2015.

[43] Open Images Dataset Github. https://github.com/cvdfoundation/open-images-dataset.

[44] Scalability and performance targets for standard storage accounts. https://learn.microsoft.com/en-us/azure/storage/common/scalability-targets-standard-account.

[45] Noah Watkins and Michael Sevilla. Using lua in the ceph distributed storage system. In *Proceedings of the Lua Workshop*, pages 16–17, 2017.

[46] Amazon S3 Object Lambda. https://aws.amazon.com/s3/features/object-lambda/.

[47] NVIDIA DALI. https://developer.nvidia.com/dali.