Technische Universität München
TUM School of Computation, Information and Technology

Performance Optimization in Deep Learning Systems

Alexander Erben

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Florian Matthes

Prüfende der Dissertation:

1. Prof. Dr. Hans-Arno Jacobsen
2. Prof. Dr. Pramod Bhatotia
3. Prof. Dr. Maarten van Steen

Die Dissertation wurde am 16.02.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 18.06.2024 angenommen.

Acknowledgments

To my wife.

Abstract

Optimizing system performance can be done in two ways. First, by buying better and more efficient hardware. Second, improving the software, particularly the underlying algorithms, can improve resource utilization and throughput. The key difference between the two approaches is that hardware investment is a one-time improvement, whereas software can be repeatedly improved. Therefore, software has a multiplicative effect by simultaneously improving performance on old and new hardware.

As the Deep Learning (DL) field continues to expand, the techniques available for training DL models are growing rapidly. While adding to the existing techniques to improve performance may seem beneficial, we suggest an empirical approach. It involves thoroughly examining each technique's assumptions, testing it in real-world environments, and measuring system-level and application-level metrics. We have found that adhering to this simple guideline is highly effective in uncovering inefficiencies and validating existing research. Following this approach can lead to the development of more efficient applications, enabling sustainable hardware utilization and improving cost efficiency. Therefore, adopting this approach is crucial for advancing the research field and enhancing our understanding of DL performance optimization.

This publication-based dissertation comprises two papers that argue for more efficient resource utilization by challenging the status quo on common DL practices and providing practical guidelines and future optimization potential. By applying an empirical approach to tackling DL bottlenecks, this work helps to improve end-to-end training times of DL systems by leveraging existing hardware focusing on both data preprocessing and DL training.

Zusammenfassung

Die Optimierung der Systemleistung kann auf zwei Arten erfolgen. Erstens durch den Kauf besserer und effizienterer Hardware. Zweitens kann durch die Verbesserung der Software, insbesondere der zugrunde liegenden Algorithmen, die Ressourcennutzung und der Durchsatz verbessert werden. Der Hauptunterschied zwischen den beiden Ansätzen besteht darin, dass die Investition in Hardware eine einmalige Verbesserung darstellt, während die Software wiederholt verbessert werden kann. Daher hat die Software einen Multiplikatoreffekt, indem sie die Leistung auf alter und neuer Hardware gleichzeitig verbessert.

Da sich das Feld des Deep Learning (DL) weiter ausbreitet, wachsen die verfügbaren Techniken für das Training von DL-Modellen schnell. Auch wenn es vorteilhaft erscheinen mag, neue Techniken zu entwickeln, um die Leistung zu verbessern, schlagen wir einen empirischen Ansatz vor. Dazu gehört eine gründliche Prüfung der Annahmen jeder Technik, das Testen in realen Umgebungen und die Messung von Metriken auf System- und Anwendungsebene. Wir haben festgestellt, dass die Einhaltung dieser einfachen Richtlinie sehr effektiv ist, wenn es darum geht, Ineffizienzen aufzudecken und bestehende Forschungsergebnisse zu validieren. Die Befolgung dieses Ansatzes kann zur Entwicklung effizienterer Anwendungen führen, die eine nachhaltige Nutzung der Hardware ermöglichen und die Kosteneffizienz verbessern. Daher ist die Anwendung dieses Ansatzes von entscheidender Bedeutung, um das Forschungsfeld voranzubringen und unser Verständnis der DL-Leistungsoptimierung zu verbessern.

Diese publikationsbasierte Dissertation umfasst zwei Arbeiten, die für eine effizientere Ressourcennutzung plädieren, indem sie den Status quo gängiger DL-Praktiken in Frage stellen und praktische Leitlinien sowie künftiges Optimierungspotenzial bieten. Durch die Anwendung eines empirischen Ansatzes zur Bewältigung von DL-Engpässen trägt diese Arbeit dazu bei, die End-to-End-Trainingszeiten von DL-Systemen zu verbessern, indem sie die vorhandene Hardware nutzt und sich sowohl auf die Datenvorverarbeitung als auch auf das DL-Training konzentriert.

Contents

1. Introduction

Efficient resource utilization is crucial in today's world [Cli22]. Driven by the environmental impact and costs, Deep Learning (DL) has been in need of performance optimization due to its popularity and widespread usage [SGM19]. The field of DL has made significant progress in increasing its utility to the general public, particularly with Large Language Models (LLMs) and generative models. This new service landscape is driven by large multi-modal models that must be cost-efficient to generate revenue [Ope22; Goo23]. At the heart of it all is the hardware utilization of underlying accelerators [Sil+23; Iva+21; Jia+20], which strive to ensure the training of larger and more powerful models to handle an ever-increasing volume of data [Hof+22].

While current research suggests that hardware and software innovations are roughly equally responsible for performance improvements, software is heavily leveraging compute-augmentic algorithmic advances rather than being more data-efficient [EB22]. For example, techniques such as quantization [Det+22] and sparse computation [DZ19] are often only enabled by specialized hardware and are typically not applicable for older accelerators as they lack the necessary underlying architecture. This emphasizes performance optimization for new hardware while deprecating older and typically more available hardware. While there is an argument to be made on energy efficiency due to new hardware typically having a better performance-to-energy ratio, the resources for manufacturing and purchasing have already been spent. As an example, the T4 GPU [Tec18], while being five architectures behind the state-of-the-art (Turing vs. Hopper), is still available on most cloud providers [EMJ23], which argues for its longevity and cost-efficiency.

This work shifts the focus of performance optimization towards improving resource utilization within DL systems, particularly with algorithmic advancement using older hardware. While aiming to increase the DL training throughput, this work challenges common DL practices by offering a new perspective on efficient hardware utilization. It provides guidelines for previously overlooked optimization potential and for leveraging existing hardware rather than upgrading it.

1.1. Motivation

In DL, hardware improvements and sheer compute quantity are big drivers for improved performance, showcased by the investment into private datacenters [MET24; Goo24; Mic23]. More specifically, current DL models have a weakness regarding their architecture, as their algorithmic intensity limits their throughput. On a single GPU, empirical evaluation has shown that many underlying DL operations are limited by data movement rather than compute capabilities [Iva+21; DAW22]. This observation also holds in distributed computing, showcased by the low compute utilization when training large DL models [Cho+23]. Additionally, the emergence of foundation models and their usefulness to the general public has led to a booming hardware market. The beneficiaries of this trend are clear: the hardware manufacturers and institutions that can afford to build data centers with the latest hardware. However, due to the lack of funds, researchers and small businesses are missing out on an emerging field and may not reap the benefits of this impactful technology. This situation leads to an obvious conflict: institutions with large budgets have no interest in facilitating easier entry into DL research, and hardware manufacturers have no interest in improving their software, which diminishes their main selling point of better performance.

As members of the scientific community and the engineering field, we are responsible for improving algorithmic development to provide fair access to technological advances and affordability in research opportunities. Additionally, we should promote the sustainable use of hardware resources, discourage large companies from dominating research fields and monopolize the benefits of DL.

1.2. Problem Statement

As the field of DL grows, the techniques applied keep increasing. For example, one of the earlier success stories in DL, AlexNet (2012), used a training regime with a pre-processing pipeline with augmentation and trained the model via Stochastic Gradient Descend (SGD) [RM51] on a single consumer-grade GPU [KSH12]. In contrast, there are many new techniques to train a DL model. For the training process, we can apply distributed training via Distributed Data Parallel (DDP) [Li+20], Fully-Sharded Data Parallel (FSDP) [Zha+23] and Hybrid-Sharded Data Parallel (HSDP) [Zha+22], memory optimized training via the Zero Redundancy Optimizer (ZeRO) [Ras+20], sparisification [DZ19], activation checkpointing [Ras+20], and quantization [Det+22]. For the gradient calculation, we have a mass of optimizers and their specific parametrization to choose from, e.g., SGD [RM51], Adam [KB14], AdamW [LH17], LAMB [You+19] and different learning rate and momentum schedulers [Goy+17; Che+21]. For general model performance, a model compilation step for both training and inference is introduced to optimize further and fuse operations together [Niu+21].

We believe an empirical approach to performance optimization may be more effective than adding to existing techniques. First, critically examining the assumptions of each technique, checking for misconceptions, and designing a real-world testbed to validate theoretical results. Second, it is important to ensure systems operate in real-world environments rather than simulated ones. Third, rigorously measuring system-level (e.g., FLOPS, memory bandwidth) and application-level metrics (e.g., throughput, data ingestion rate). Based on our experience, adhering to these simple guidelines is bound to reveal inefficiencies and confirm existing research, *both* of which are crucial for advancing the research field and enhancing our understanding. By following this approach, we can create more efficient applications, enable sustainable hardware utilization, and improve cost-efficiency.

This work follows these guidelines in the context of end-to-end DL performance optimization.

1.3. Contributions

This publication-based dissertation comprises two papers that argue for more efficient resource utilization by challenging the status quo on common DL practices and providing practical guidelines and future optimization potential. By applying an empirical approach to tackling DL bottlenecks, this work helps to improve end-to-end training times of DL systems while using typically older and more available hardware. The main contributions of this work are:

1. We present a study demonstrating how the intermediate materialization strategies of preprocessing pipelines can significantly impact throughput. This highlights the need to reevaluate pipeline optimization strategies, as the commonly used "fully offline" and "fully online" preprocessing strategies may not be the most efficient options. To demonstrate the generalizability of our findings, we tested our claims on four DL domains and evaluated seven representative pipelines under different setups of caching, compression, and parallelization capabilities.

2. We provide a new, cost-efficient approach for training DL models across multiple continents and clouds, utilizing a decentralized middleware initially created to enable collaborative DL training. By leveraging the global spot pricing market, we demonstrate that we can improve performance and be more cost-efficient by employing several low-cost VMs over multiple continents instead of more centralized and powerful hardware. With the granularity metric, the ratio between computation and communication, we demonstrate how to evaluate the suitability of models for distributed training under low bandwidths. Finally, to provide guidance on using this training method in the real world, we argue how to leverage different pricing structures of cloud providers for maximum cost-efficiency.

1.4. Outline

We begin this work by introducing our goals, the high-level motivation, and a concrete problem statement. Then, we provide a brief introduction to the recent hardware and software improvements, including popular techniques and the current challenges of improving performance with different distributed processing approaches. The publications are then summarized to showcase our research contributions. Finally, we conclude by discussing our contributions on a grander scheme.

Although I am the sole author of this dissertation, I use the plural pronoun "we" throughout the manuscript to convey that my understanding is a result of the effort of the global research community, my colleagues from our research group, and my collaborators.

Parts of the content and contributions of this work have been published in:

A. Erben*, R. Mayer, J. Jedele, and H.-A. Jacobsen. "Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines." In: *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD '22. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 1825–1839. ISBN: 9781450392495. DOI: 10.1145/3514221.3517848 (*formerly Alexander Isenko)

A. Erben, R. Mayer, and H.-A. Jacobsen. *How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study*. 2023. arXiv: 2306.03163 [cs.LG]

2. Methodology

This chapter gives an overview of the hardware and algorithmic progress since the rise in popularity of DL, combined with relevant techniques that pushed the envelope of DL performance optimization. As we already outlined our methodology in Section 1.2, we highlight specific techniques relevant to understanding our approach and the current DL performance optimization landscape. Section 2.1 explains the effect of DL on hardware evolution and why the current progress only benefits specific optimization techniques, further motivating our work. Section 2.2 presents key optimization techniques that allow for the democratization of DL models by enabling the training of larger model sizes and better utilization of the underlying hardware. Finally, Section 2.3 covers the different ways to distribute the DL training process over multiple GPUs as it is a necessary part of current training regimes.

2.1. Deep Learning Hardware Evolution

A few key metrics are important to understand how DL has gained popularity in recent years. We show the evolution of NVIDIA GPUs since 2011, just before the publication of AlexNet, with memory capactiy (Figure 2.1), floating point opertaions per second (FLOPS) (Figure 2.2), memory bandwidth (Figure 2.3) and FLOPS to memory bandwidth ratio (Figure 2.4).

We included some of the most popular GPUs from the consumer class (GeForce and TITAN), the most powerful workstation GPUs (Quadro), and the most popular data center GPUs (K40, T4, P100, V100, A100, H100) to get a feel for how the market developed over time [Wik24]. All figures have three key markers included. First, AlexNet in 2012, arguably a key milestone for the CV community due to its significant performance on the ILSVRC2012 dataset [KSH12]. Second, BERT in 2018, a bi-directional, transformer-based NLP model that showcased the importance of different pre-training tasks and reached state-of-the-art performance on GLUE, MultiNLI and SQuAD [Dev+18]. Finally, the release of GPT-3 in 2020 by OpenAI, a decoder-only transformer-based model with 175B parameters, which showed very strong resonance with the general public regarding its usefulness [Bro+20]. This, and subsequent releases by OpenAI generated additional interest in the hardware market.

Generally, all of the metrics we compare show exponential growth. However, some do so more than others, which we want to showcase here.

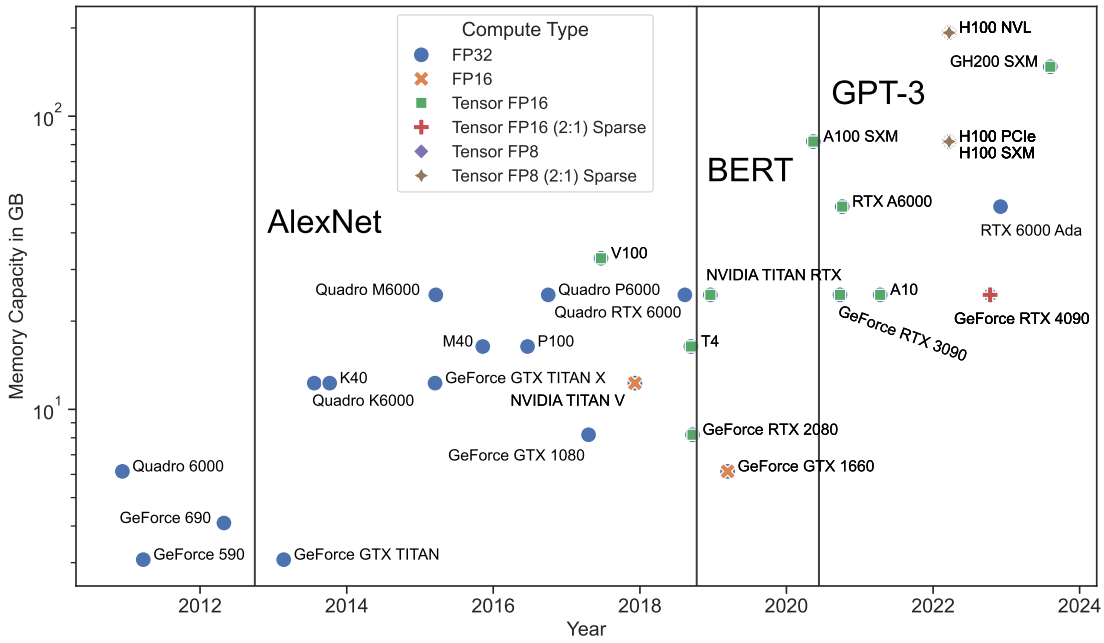


Figure 2.1.: Memory capacity in GB over time of NVIDIA GPUs on a logarithmic scale. A linear trend is visible, meaning a roughly exponential improvement over time. Until BERT, all market segments were close to each other, but after BERT, the consumer GPUs are the only ones stuck at 24 GB memory (GeForce 4090), while workstations reach 48 GB (RTX A6000) and datacenter GPUs reach 147 GB (GH200 SXM).

The memory capacity of GPUs is increasing, although not as much as other components (Figure 2.1). The best consumer GPUs are only allowed up to 24 GB (GeForce 4090), while workstation GPUs have a capacity of up to 48 GB (RTX 6000 Ada). The latest two releases of datacenter GPUs (A100 and H100) have 80 GB high-bandwidth memory and are specifically designed for DL workloads. Even though memory capacity is a significant limitation for the size of models that can be trained, we do not observe extreme growth. The reasons for this is explained in the following paragraphs.

The number of FLOPS in DL hardware has steadily increased (Figure 2.2). Before 2017, FLOPS meant computation only in FP32, meaning that every floating point value

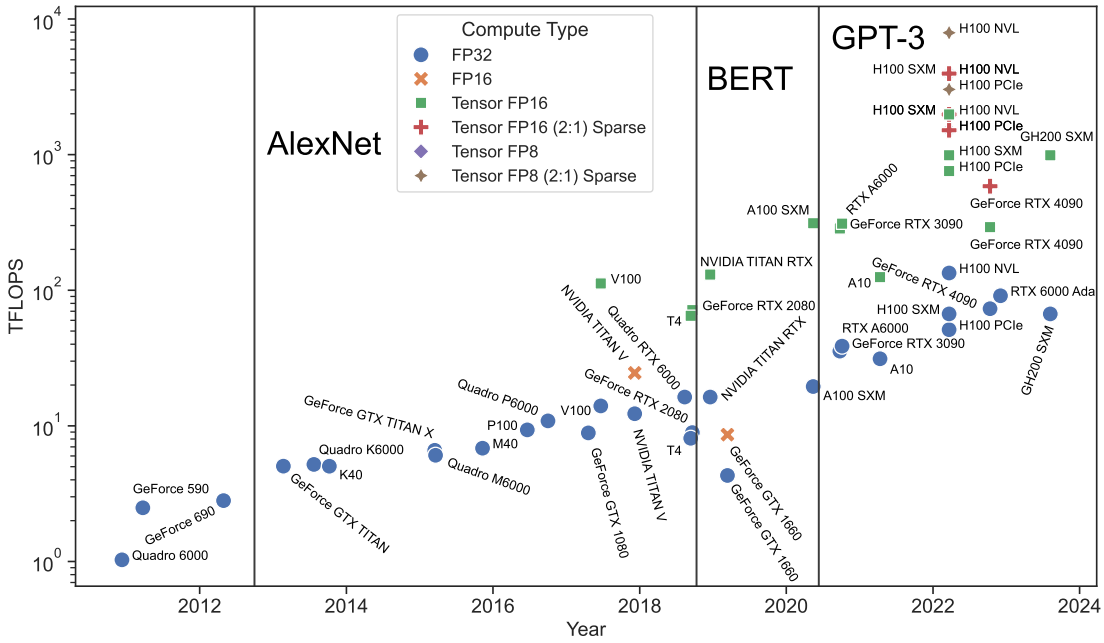


Figure 2.2.: TFLOPS over time of NVIDIA GPUs on a logarithmic scale. Until BERT, a slow linear trend is visible, but with the advent of Tensor Cores and the Tensor FP16 datatype, an almost double exponential trend is visible. While the FP32 throughput slowly increases on the same trajectory, smaller datatypes push compute capabilities by two orders of magnitudes (H100 NVL FP32 vs. Tensor FP8 (2:1 Sparse)).

is stored in 32 bits. However, researchers have discovered that training with quantization [Det+22], meaning fewer bits, was not affecting performance significantly, so NVIDIA invested in the first GPU with dedicated FP16 compute capabilities (V100), which had 112 TFLOPS. This was achieved by incorporating dedicated Tensor Cores designed for General Matrix Multiplication (GEMM), a key operation in DL models. In 2018, even consumer graphics cards started getting Tensor Cores with the GeForce RTX 2080, while workstation GPUs received Tensor Core support in 2020 with the RTXA6000.

Following this, there has been an exponential but slow improvement in FP32 computational capability, reaching the same Tensor FP16 performance of 2017 (H100 NVL vs. V100). However, starting in 2022, a new optimization potential has emerged in the form of structured sparsity. The Ampere architecture was the first to provide some operations to support this, and the newest Hopper architecture now has full-fledged support for a 2:4 sparsity pattern (where 2 out of 4 sequential values are zero). This

effectively doubles the throughput, as half of the input matrices are empty, resulting in no less than 3958 Tensor FP16 TFLOPS (H100 NVL).

Both quantization and sparsity show many numerical opportunities for performance optimization within DL training. Despite losing roughly half the precision and values, models still perform well [DZ19]. The newest GPUs, the H100, even support FP8 and INT8, which further reduces the storage consumption of floating point and integer representations.

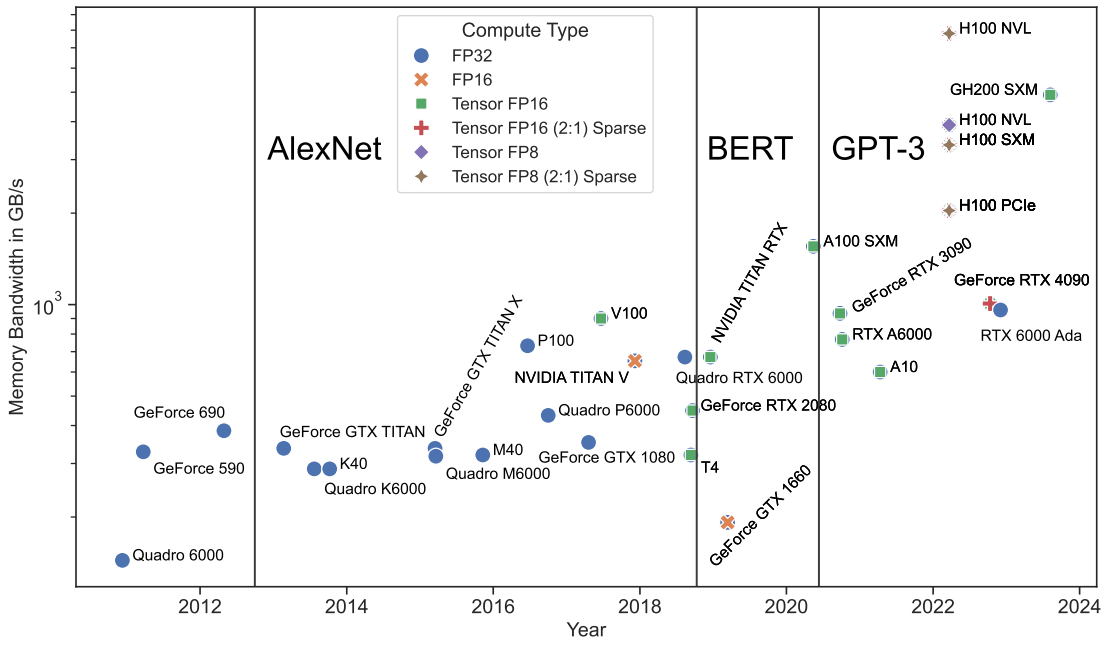


Figure 2.3.: Memory bandwidth in GB/s over time of NVIDIA GPUs on a logarithmic scale. Until GPT-3, a mostly linear trend is visible. With the popularity of transformer architecture rising, memory bandwidth became an even higher priority and improved by almost one order of magnitude (A100 SXM vs. GH200 SXM).

While FLOPS were, and still are improving at a close to double exponential rate due to recent advances in numerical stability and hardware manufacturing, memory bandwidth has been increasing on a similar scale (Figure 2.3). This is because, currently, at least half of the training time is spent on memory movement rather than computation [Iva+21]. Many DL operations, such as activations and element-wise operations

(e.g., vector addition), have a low arithmetic intensity, a ratio showing how much data has to be moved compared to the amount of computing needed [Ofe+14].

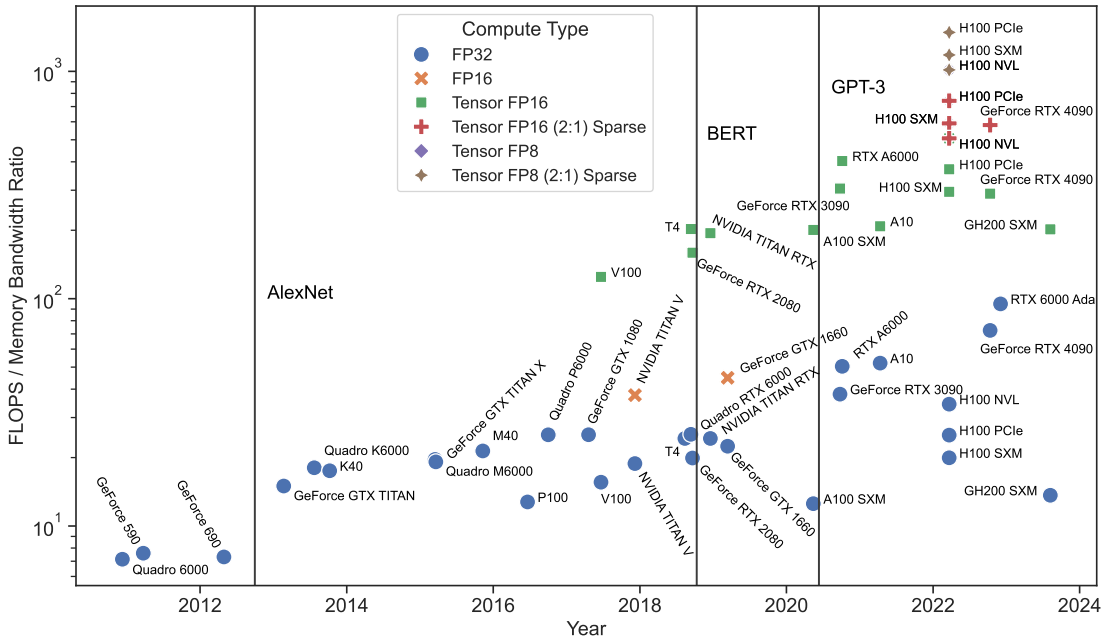


Figure 2.4.: FLOPS to memory bandwidth ratio over time of NVIDIA GPUs on a logarithmic scale. While trends are visible, they are divided into the respective compute types. FP32 is slowly improving on the same trajectory, with a slight bias towards more memory bandwidth. Tensor FP16 is stalling due to high investment into higher bandwidths rather than compute capabilities. However, sparsity-based types play in their own league as they benefit both compute and memory bandwidth capabilities.

GPUs, or any specialized accelerators, are specifically designed for GEMMs, as having a runtime complexity of $\mathcal{O}(n^2)$ which means that for every parameter n, we are performing n^2 computations and $\mathcal{O}(n)$ data movement. Unfortunately, many DL training operations perform much smaller ratio, leading to memory bandwidth bottlenecks [Jia+20]. As this obstacle is not trivial to circumvent algorithmically, heavy investment into memory bandwidth was channeled, reaching currently up to 3.9 TB/s and 7.8 TB/s with sparsity (H100 NVL). Arguably, with the meteoric improvement of FLOPS, this is currently the largest bottleneck regarding DL training and inference performance. To fully understand the gravity of how imbalanced both metrics have risen while being on an exponential trend, Figure 2.4 shows the ratio of FLOPS divided by

the memory bandwidth. If we disregard the current potentially "lossy" computational advancement with sparsity, the FLOPS to memory bandwidth ratio has slowed down significantly in growth with Tensor FP16. Even consumer-level GPUs like the RTX 2080 from 2019 are close to the A100 and the newest chip, GH200 SXM.

The recent hardware upgrades seem to align with the significant points mentioned in the figures, indicating that the bottlenecks and scalability issues described in this section are not entirely new. While we only discussed NVIDIA-based hardware due to its ubiquity and being one of the key players in the rise of DL, we have omitted other companies also developing custom deep-learning chips. Although most of them are still in the early stages and typically not available to the general public, it is evident that hardware improvements are driving DL performance [IBM23; Goo15; Cer24; Int23]. As it currently stands, there will be ample competition in this market, further driving hardware improvements.

Preparing for this future and simultaneously alleviating the memory bandwidth issues can be done by a model compilation step, which we discuss in the next section.

2.2. Algorithmic Runtime Improvements

While researchers can rarely affect the design decisions of hardware manufacturers, there are multiple ways in which algorithmic design decisions can utilize hardware more efficiently. We cover some of the more popular techniques currently employed to reduce memory pressure and improve hardware utilization.

Model Compilation

PyTorch model code is usually translated to C++ to call the CUDA runtime, enabling the use of an NVIDIA GPU. However, developers of deep learning frameworks face the challenge of balancing generalizability and performance optimization [GRK17]. They must choose between manually optimizing performance for every single case by writing low-level code or using general-purpose operations provided by vendor libraries such as cuDNN. This can be time-consuming and not easily portable. Moreover, as described earlier, some operations commonly used in deep learning, such as activation functions, have a low arithmetic intensity and suffer from memory-bandwidth bottlenecks.

One solution to these problems is using a model compilation step, which can serve as an intermediate language representation between PyTorch and vendor libraries. It can also perform operator fusion due to static program analysis, allowing it to effectively

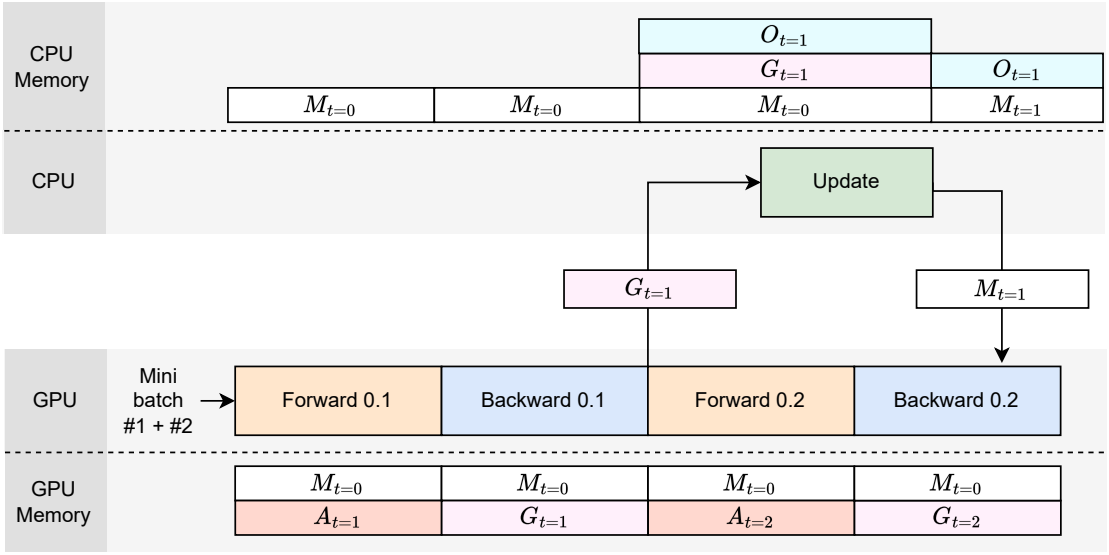


Figure 2.5.: ZeRO-Offload training process with memory utilization. Legend: M = model, A = activations, G = gradients, O = optimizer states (e.g., momentum). The forward and backward steps are enumerated by "X.Y", where X is the model timestep, and Y is the minibatch counter. By utilizing the Delayed Parameter Updates technique, the model update is performed on the CPU and reduces the memory pressure of keeping the entire optimizer state on the GPU.

execute functions without accessing memory due to their sequential nature.

PyTorch features like `torch.compile` [PyT23] fill this gap by using the Triton compiler [TKC19], which currently addresses some of the challenges regarding memory bandwidth bottlenecks. While we did not employ these relatively new techniques in our work, we are hopeful that they will enable building specialized code to run on older accelerators much more efficiently.

ZeRO-Offload

Due to the rise of large transformer-based models in recent years [Sho+19; Hof+22], the model states, consisting of the gradients, parameters, and optimizer states that are kept in GPU memory, are becoming major bottlenecks for increasing model sizes.

To provide easier access to large DL models, one solution is to include the CPU and the random-access memory (RAM) in the DL training process, called ZeRO-

Offload [Ren+21]. We showcase the training process by incrementally going through it step by step (Figure 2.5). During the forward and backward pass, the activations and gradients are kept on the GPU as usual. However, when the update step is triggered, the gradients are moved from GPU memory to CPU memory, with all the remaining optimizer states (e.g., momentum, variance) and the model itself already there. So that there is no idle time for the GPU, it simply trains further with the next minibatch and a non-updated model at $t = 0$. This technique is called Delayed Parameter Updates (DPU), and while changing the training process, it seems not to affect convergence when introduced after a few iterations. This allows us to interleave GPU and CPU computation to maximize their respective strengths. The CPU typically has a much larger memory, making it easier to store model parameters and optimizer states, while the GPU performs the fastest with the computationally intensive steps of the forward and backward pass. Finally, after the CPU has finished with the update, it is typically faster than the next $t = 1$ step on the GPU, so the $t = 1$ model and $t = 2$ gradients can be exchanged.

By using ZeRO-Offload, one can enable much larger minibatch sizes and larger models, as only the activations and gradients are stored in an intermediate way on the GPU memory without keeping the entire optimizer state. This democratizes DL applications by allowing people to utilize their existing and powerful hardware in a way that allows them to increase their model sizes by up to $10\times$ on a single GPU. The DPU technique employed here is a key component of the decentralized framework we used in our contributions, enabling the overlapping of computation and communication when training over long distances.

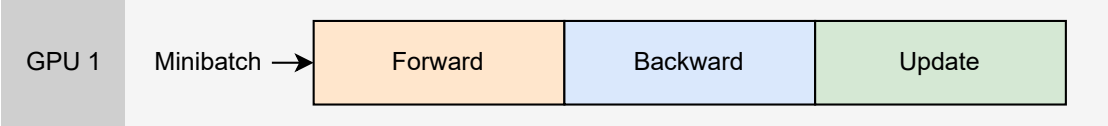


Figure 2.6.: Single GPU training process. Neither training nor model data leaves the device.

2.3. Distributed Deep Learning

There are many ways to improve DL training time by utilizing additional hardware. A popular abbreviation is *3D-parallelism*, which stands for the three types of parallelism that can be used: model, data, and pipeline parallelism. This section describes each of them, their advantages, disadvantages, and their scalability potential. We use a Graphics Processing Unit (GPU) as an example of an accelerator due to its ubiquity. However, it can be replaced by any processing unit that implements a fast general matrix multiplication, e.g., Tensor Processing Units (TPU) [Goo15].

To set a baseline, Figure 2.6 shows a single GPU model training setup, which consumed a minibatch of varied size, performs a forward pass, calculates the loss (not shown due to negligible computation time), calculates the gradients based off of the loss and activations in the backward step and updates the weights w.r.t. the gradients. This process can be repeated as often as needed until a desired loss is reached. A limiting characteristic of this process is the sequential nature of the steps, i.e., one can only interleave the forward and backward pass calculation for the same data point by changing the algorithmic basis. Moreover, this operates typically at near maximum memory capacity and is limited due to memory bandwidth, as there are benefits to larger model sizes for model accuracy [Hof+22].

Data Parallelism

Data parallelism is the simplest type of parallelism that stores the same model on all GPUs and performs the same training cycle with different minibatches (Figure 2.7). To keep the model consistent between GPUs, the calculated gradients after the backward pass are summarized via an AllReduce operation. This allows the update step to happen in the same fashion as in the single GPU scenario. An example of this is PyTorch DDP [Li+20].

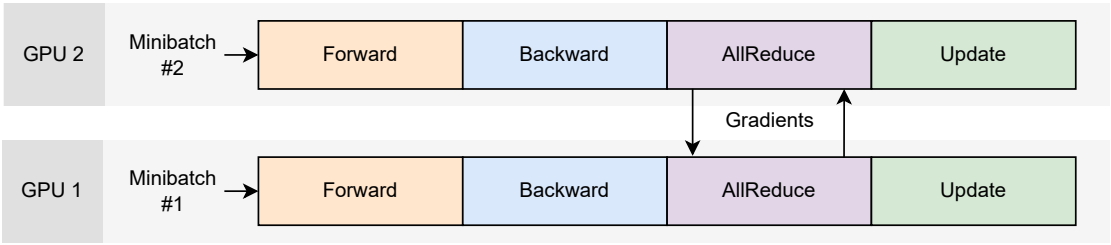


Figure 2.7.: Data parallelism on two GPUs. Each GPU consumes their own data and averages the gradients from both minibatches to keep the model consistent over training steps.

Advantages:

- It is typically trivial to use and implement without major architectural requirements from the model or hardware.
- The throughput improvement potential is high because it is a trivially parallelizable process with a single barrier after the backward pass.

Disadvantages:

- No inherent support for heterogeneous hardware in compute and network connectivity, as the AllReduce call will be waiting for the slowest GPU.
- Can only be leveraged if the optimizer and model work with the increased minibatch size. If the original minibatch size is divided, the speedup can be limited due to a larger communication time.
- It can only trivially be applied to models that fit onto a single GPU. While it is possible to utilize model and pipeline parallelism, it has compounding negative effects regarding its minibatch size and further scalability.
- Without specialized techniques, the inter-GPU bandwidth must be very large to show positive throughput effects, e.g., $> 200\,\mathrm{GB/s}$.

Scalability potential: Adding more GPUs will scale well until the communication time between the GPUs outweighs the calculation time benefits. Typically, the smaller the model parameter count, the fewer GPUs can be added in this fashion. Network bandwidth between nodes is the most limiting factor when going beyond a single multi-GPU

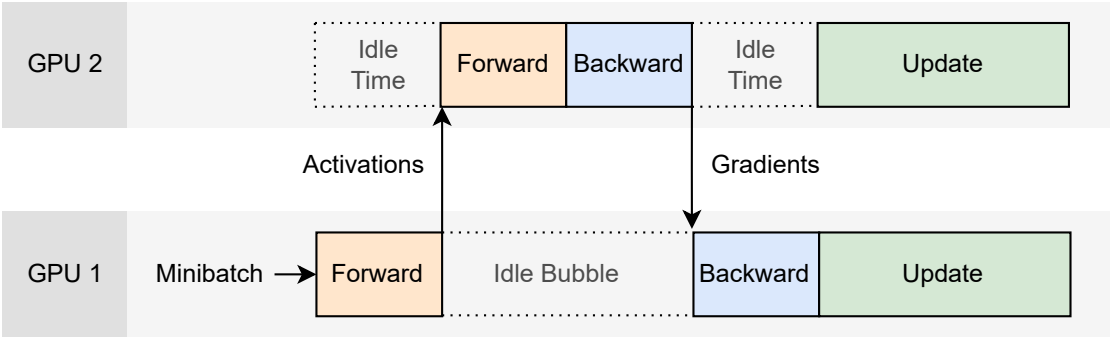


Figure 2.8.: Inter-layer parallelism on two GPUs. Only a single minibatch is consumed and the activations is forwarded between GPUs until it fully completes the model and starts backpropagation, reversing the data communication with gradients. Then, all GPUs have their respective gradients and can update to keep the model consistent.

node (e.g., DGX-3) and assuming the setup is suited for further parallelization. Cable lengths between nodes and racks will increase and drop the bandwidth significantly to, typically, 10-200 Gb/s.

Model Parallelism

Model parallelism can be performed in two ways: inter-layer and intra-layer.

Inter-layer model parallelism splits the model into multiple parts by the layer order, e.g., GPU 1 stores the model layer-[0,1] and GPU 2 stores layer-[2,3] (Figure 2.8). By the sequential nature of the training process, almost no throughput improvements can be gained with this technique, as GPU 1 has to finish processing fully before GPU 2 can start. The only performance benefit is the parallel update step on both GPUs, which, however, is typically overshadowed by the communication times between the GPUs. This parallelization technique aims to train larger models that do not completely fit onto a single GPU. As the model is split into parts (and with techniques such as ZeRO [Ras+20], so is the optimizer state), the memory utilization is reduced by the same ratio.

Advantages:

- Enabling the training of larger model sizes is essential to improving DL results.

- It enables larger minibatch sizes due to the reduced memory pressure, which creates more flexibility around optimizer parametrization and potentially faster training times.

Disadvantages:

- The decision as to how to split the model is non-trivial, as models are not typically easily dividable into N equally computationally expensive chunks. Due to this restriction, it is typically only applicable to homogeneous devices as this removes hardware capabilities from the equation.
- Without exceptionally good interconnectivity between the GPUs, this can significantly reduce throughput.

Scalability potential: Just adding additional computing capabilities will reduce throughput due to increased communication time without changing the training process. However, if a higher minibatch size is enabled by applying intra-layer parallelism, there is a potential for an improved end-to-end performance. Specifically, this can have large effects on throughput under the guidance of recent work that focuses on large minibatch size training [Goy+17]. Additionally, the model performance can increase significantly by enabling larger model sizes.

Intra-layer model parallelism (also called *tensor* parallelism) splits the model into multiple parts on a layer-basis, e.g., for all layers, GPU 1 stores the columns [0,N] from a single layer and GPU 2 stores the columns [$N+1$,M] (Figure 2.9). N is the split point (or multiple points) and is determined by the number of GPUs the model is being split between, and M is the total number of columns of the weight matrix of each layer. By dividing the work on a per-layer basis and using the same minibatch as input, the outputs of each partial layer computation must be summarized with an AllReduce operation to be functionally identical to single GPU training. The same process of communicating and processing is applied for the backpropagation step until it finishes, and the gradients can be locally updated to the layer partitions. Typically, this parallelization strategy is used in addition to sharding the activations and optimizer state to reduce duplicate information on individual GPUs' high-bandwidth memory. Examples of these are Fully-Sharded Data Parallel [Zha+23], Hybrid-Sharded Data Parallel (HSDP) [Zha+22], and ZeRO-3 [Ras+20].

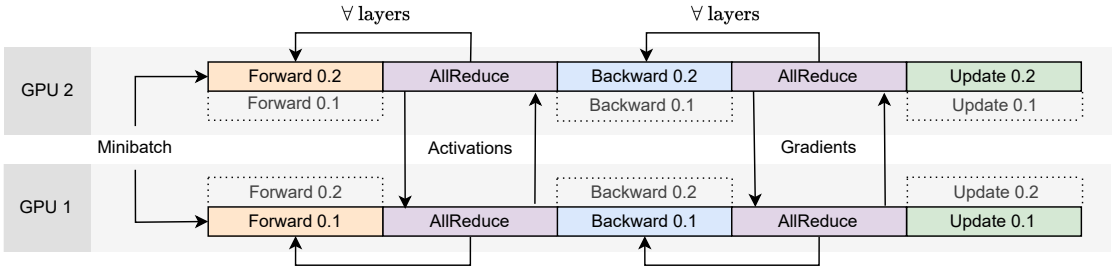


Figure 2.9.: Tensor parallelism on two GPUs. The forward and backward steps are enumerated by "X.Y", where X is the model timestep, and Y is the mini-batch counter. Splitting the model horizontally allows each GPU to keep part of each model layer. The reduced parameter count reduces memory pressure, and consistency is guaranteed by communicating the activations and gradients after each layer, respectively.

Advantages:

- Enables large models to be trained due to reduced memory pressure.
- It is easier to split into computationally equal shards by using layer columns instead of layers themselves.

Disadvantages:

- It is hard to understand due to interleaved, fine-grained communication and computation steps, which opens up scheduling and memory management issues.
- Similar to inter-layer parallelism, it also requires very good interconnectivity due to increased communication requirements.
- Even with easier sharding strategies, it is not trivial to support heterogeneous hardware.

Scalability potential: Compared to inter-layer parallelism, it enables better throughput improvements through easier sharding strategies. As communication still plays a big role in enabling this type of parallelization, the current trend moved to use HSDP to perform tensor parallelism on an intra-node setting (e.g., 800 GB/s interconnects) and data parallelism on inter-node (e.g., 200 Gb/s interconnects). While it is currently state-of-the-art in training DL models, it is heavily based on hardware capabilities and is still bottlenecked by memory and network bandwidth.

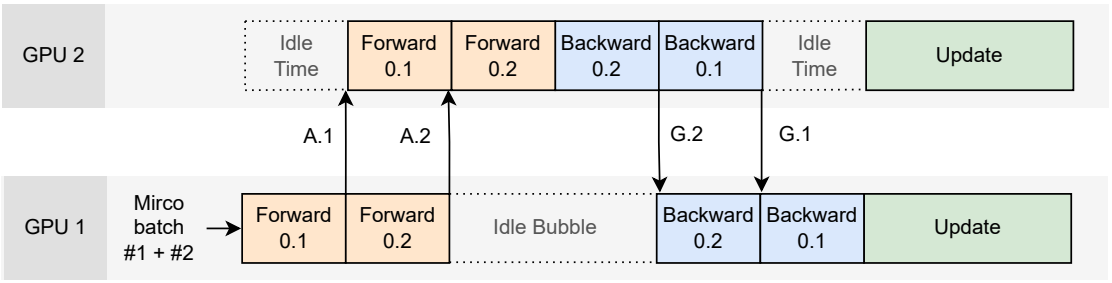


Figure 2.10.: Pipeline with inter-layer parallelism on two GPUs. The forward and backward steps are enumerated by "X.Y", where X is the model timestep, and Y is the microbatch counter. By utilizing inter-layer parallelism and splitting the minibatch into microbatches, the amount of work which leads to a lower latency with GPU 1, reducing the idle-time of GPU 2. However, without additional techniques, the idle-time of GPU 1 is increasing, also called an "idle bubble".

Pipeline Parallelism

The idea of pipeline parallelism is derived from many kinds of pipelines we encounter, such as instruction pipelines in computer architecture and assembly lines in factories. If work can be divided into smaller parts, the latency of the pipeline can be improved as the idle time of all workers is minimized. This can be applied to a model training process with inter-layer parallelism by splitting the minibatch into microbatches (Figure 2.10). While the idle time of GPU 2 is being reduced, an idle bubble still exists while GPU 1 waits on GPU 2 to finish processing the last micro-batch. To reduce this idle time, there are multiple techniques from PipeDream [Nar+19], GPipe [Hua+19], and others [Nar+21], e.g., by allowing a model from timestep $t - 2$ to run inference on a micro-batch of timestep t which removes the idle bubble completely.

Advantages:

- Improves throughput, and, combined with other types of parallelisms, can potentially combat their disadvantages.
- Enables larger model sizes to be trained using additional hardware while improving throughput compared to naive inter-layer parallelism.
- Communication load is typically lower than intra-layer parallelism.

Disadvantages:

- Minibatches have to be splittable into microbatches. This may be hard to get right due to the parametrization of the training run.
- Ramp-up and -down phases are necessary until a stable GPU utilization is reached and is limiting throughput at those times.
- Communication load is typically higher than data parallelism.

Scalability potential: Scalability is ensured as long as the model is splittable into arbitrarily small shards and the minibatch similarly into micro-batches. However, only the model's size is scaled, not its training speed. While one can argue about the improved performance of larger models on certain tasks, it is not suited for improving throughput while staying on the same hardware without modifying the underlying training algorithms.

3. Summary of Publications

This publication-based dissertation comprises the following publications:

Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines

A. Erben*, R. Mayer, J. Jedele, and H.-A. Jacobsen. "Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines." In: *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD '22. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 1825–1839. ISBN: 9781450392495. DOI: 10.1145/3514221.3517848 (*formerly Alexander Isenko)

How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study

This paper is accepted as peer-reviewed conference paper at the *Proceedings of the VLDB 2024, Volume 17* and currently in the process of being published.

A. Erben, R. Mayer, and H.-A. Jacobsen. *How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study*. 2023. arXiv: 2306.03163 [cs.LG]

3.1. Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines

This chapter is published as a peer-reviewed conference paper at ACM SIGMOD 2022 with the Artifacts Available, Artifacts Evaluated and Results Reproduced badges.

A. Erben*, R. Mayer, J. Jedele, and H.-A. Jacobsen. "Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines." In: *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD '22. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 1825–1839. ISBN: 9781450392495. DOI: 10.1145/3514221.3517848 (*formerly Alexander Isenko)

Full text version enclosed: Appendix A

Artifacts: `https://github.com/cirquit/presto`

Synopsis: Improving preprocessing performance is paramount when GPU resources are costly and rare. Increasing data quantities make this an even more pressing issue, as they introduce high storage consumption costs and are not trivial to manage in a distributed environment.

By using intermediate strategies that materialize the preprocessing pipeline partially, we are able to showcase how common preprocessing practices, e.g., full materialization, are not the best-performing approach. Additionally, we survey the optimization potential for seven representative preprocessing pipelines from four domains (computer vision, natural language processing, non-intrusive load monitoring, and automatic speech recognition), covering compression, parallelization effectiveness, and different caching approaches. Finally, we provide a system, **PRE**processing **S**trategy **O**ptimizer (PRESTO), which can automatically decide on materialization strategy for a given pipeline and dataset based on our insights and profiles.

Contributions: The thesis author implemented and developed the approach. They executed the experiments, conducted the analysis and extracted general insights. Finally, they wrote the paper.

3.2. How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study

This chapter is accepted as peer-reviewed conference paper at the *Proceedings of the VLDB 2024, Volume 17* and currently in the process of being published.

A. Erben, R. Mayer, and H.-A. Jacobsen. *How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study*. 2023. arXiv: `2306.03163 [cs.LG]`

Full text version enclosed: Appendix B

Artifacts: `https://github.com/cirquit/hivemind-multi-cloud`

Synopsis: Decentralized systems provide an almost unlimited potential for scalability and enable the democratization of applications. Within Deep Learning, this was unthinkable due to current frameworks being unable to cope with interruptable peers and needing homogeneous, high-performance hardware. This is becoming a reality with Hivemind, the first-of-its-kind collaborative DL framework.

By utilizing Hivemind, we showcase how spot instances distributed over the entire globe and belonging to different cloud providers can be leveraged to be both faster and cheaper compared to more centralized hardware setups. We establish a lower bound on model sizes suitable for training over continents in low-bandwidth (< 1 Gb/s) and high-latency (> 150 ms) environments and provide guidance on estimating a model's scalability potential. By performing real-world experiments on three cloud providers and four continents, we prove that leveraging the global spot price market for cost-efficient multi-, hybrid-cloud, and geo-distributed DL training is possible.

Contributions: The thesis author conceived, implemented and developed the approach. They executed the experiments, conducted the analysis and extracted general insights. Finally, they wrote the paper.

4. Discussion

This chapter discusses our results in the larger context of DL performance optimizations. By pursuing an empirical approach to find optimization potential, we assess its potential and review related approaches. We conclude this chapter by showcasing that while current performance optimization techniques have their place, they do not replace our algorithmic-centric view, which offers additional benefits for accessibility to the DL field.

Our work on preprocessing pipelines showcases different tradeoffs in multiple DL domains and highlights the importance of profiling rather than applying best practices. Even with storage layers that enable multi-threaded read access and allow for $> 10\,\mathrm{Gb/s}$ read speeds, we still saw a surprising impact of storage consumption on all parts of the preprocessing pipeline. The multi-threading capabilities and the final data provisioning rate of the storage hardware, the interconnects, the deserialization process, and the pipeline itself were massively affected by an increase and decrease in total storage consumption. While utilizing multiple workers to process the data in parallel seems like an obvious choice for a trivially parallelizable problem, it is not a silver bullet. We encountered various issues, from significant overheads of thread initialization due to small sample sizes to native and unoptimized implementations that do show speedup but fall short to single-threaded third-party implementations. Finally, by leveraging general-purpose compression, which is not specifically designed for de-/compression speeds or maximum space-saving, we found performance potential under some circumstances.

Many studies have dealt with performance optimization within data provisioning [Kak+19; Zhu+18; Moh+20; Kan+20]. Our work stands out due to our initial set of assumptions. We utilize consumer-grade hardware without access to abundant amounts of memory, which may absolve any intermediate materialization strategy by caching the entire dataset in memory. Additionally, we do not use accelerators to offload the preprocessing steps, which creates higher memory pressure on an already scarce resource while performing DL training. Finally, we kept the pipelines functionally consistent and did not employ techniques that modified the data in any way, making them more generally applicable.

Our work on decentralized DL training over multiple continents and different clouds provides a novel contribution to the field of DL training that enables a new, cost-efficient way of training models on spot instances. We provide a first step into making distributed DL training geographically scalable [VT16]. While there is related work on decentralized Stochastic Gradient Descend [Lia+17], training with unreliable peers in a collaborative setting [Dis+21] and training large models via model and pipeline parallelism in a decentralized setting [Rya+23; Yua+22], we are the first to show that relatively small models can still benefit from this approach. The intuition behind training in low-bandwidth and high-latency environments is that larger models are much easier to train efficiently due to their high computational requirements. With a technique like Delayed Parameter Updates [Ren+21], the computation and communication steps can be overlapped, allowing longer communication times if computation time increases. However, due to the square-cube law [Rya+23], decreasing the model parameter count reduces the computational complexity in a quadratic fashion (due to matrix multiplication being a $\mathcal{O}(n^2)$ operation, and the communication complexity in a linear fashion ($\mathcal{O}(n)$). We define the lower bounds of model sizes that can still be efficiently trained in these environments by utilizing the granularity metric, providing a reference evaluation from training on four continents, and showing the scalability potential of structuring such a training run most cost-efficiently. As training foundation models is (at least currently) not a common nor cheap task, we enable users to leverage competing cloud providers and their spot pricing structures to train more specialized, smaller models and provide a cheaper entry to the benefits of the field of DL. With this work, we provide one of the building blocks to move DL training to the domain of Sky Computing, specifically combined with cloud broker systems like SkyPilot [Yan+23], initially proposed by the work on Conductor [Wie+10].

5. Concluding Remarks

In order to improve resource utilization and end-to-end DL training time, this work presented optimization potential in both data preprocessing pipelines and DL model training. The ideas provided in this work stem from an empirical approach to tackling performance optimization within DL. By utilizing problem settings inspired by actual real-world usage and evaluating our techniques on a wide range of domains, we showcase their applicability and resilience to the fast-paced DL research field.

With techniques to find the optimal materialization strategy for preprocessing pipelines, we have helped to improve hardware utilization as the DL training ingestion rate can be limited by preprocessing throughput. We kept the pipeline functionally identical and showcased the tradeoff between storage consumption and throughput. Additionally, we evaluated general-purpose techniques, such as compression and caching, on their applicability to multiple domains.

By utilizing a collaborative DL framework to enable multi- and hybrid-cloud training on spot instances over different cloud providers, we provide the basis for DL training to step into Sky Computing. We fill the gap of understanding as to when models are suited for training under such network-constrained situations and show that even relatively small models still experience an improvement in throughput when trained in this fashion. Our work shows that training over multiple continents is possible, and by utilizing brokers between cloud providers with support for spot VMs [Yan+23], we open the possibility to enable auto-migrated, decentralized DL training for the best spot prices in the world. This helps both the cloud providers, as they can provide a new service with older hardware, and the users, as they have a new possibility to choose much cheaper hardware at varying training speeds. By having the ability to avoid vendor lock-in with different cloud providers, being resilient to interruptions, and having the option to migrate while training leads to increased competition and lower prices.

Appendices

A. Where Is My Training Bottlenck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines

Reprinted with permission from

A. Erben*, R. Mayer, J. Jedele, and H.-A. Jacobsen. "Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines." In: *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD '22. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 1825–1839. ISBN: 9781450392495. DOI: 10.1145/3514221.3517848 (*formerly Alexander Isenko)

Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines

Alexander Isenko, Ruben Mayer, Jeffrey Jedele
Technical University of Munich, Germany
{alex.isenko},{ruben.mayer},{jeffrey.jedele}@tum.de

Hans-Arno Jacobsen
University of Toronto, Canada
jacobsen@eecg.toronto.edu

ABSTRACT

Preprocessing pipelines in deep learning aim to provide sufficient data throughput to keep the training processes busy. Maximizing resource utilization is becoming more challenging as the throughput of training processes increases with hardware innovations (e.g., faster GPUs, TPUs, and inter-connects) and advanced parallelization techniques that yield better scalability. At the same time, the amount of training data needed in order to train increasingly complex models is growing. As a consequence of this development, data preprocessing and provisioning are becoming a severe bottleneck in end-to-end deep learning pipelines.

In this paper, we provide an in-depth analysis of data preprocessing pipelines from four different machine learning domains. We introduce a new perspective on efficiently preparing datasets for end-to-end deep learning pipelines and extract individual trade-offs to optimize throughput, preprocessing time, and storage consumption. Additionally, we provide an open-source profiling library that can automatically decide on a suitable preprocessing strategy to maximize throughput. By applying our generated insights to real-world use-cases, we obtain an increased throughput of 3× to 13× compared to an untuned system while keeping the pipeline functionally identical. These findings show the enormous potential of data pipeline tuning.

CCS CONCEPTS

• **Information systems** → **Extraction, transformation and loading**; *Data management systems*; Storage management; • **Hardware** → *External storage*; • **General and reference** → *Performance*;

KEYWORDS

preprocessing, data processing, datasets, machine learning, deep learning

ACM Reference Format:
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 (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3514221.3517848

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA.
© 2022 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517848

1 INTRODUCTION

Deep learning (DL) models are used in multiple areas, ranging from e-mail spam filtering [10] in natural language processing (NLP) to image segmentation tasks for autonomous driving [33] in computer vision (CV). The improvement of these models is not only based on more advanced architectures or algorithms [50, 80, 86], but also on increased quality and quantity of training data [16, 23, 48, 53, 80]. Having more training data usually proves to be beneficial to the model performance [7].

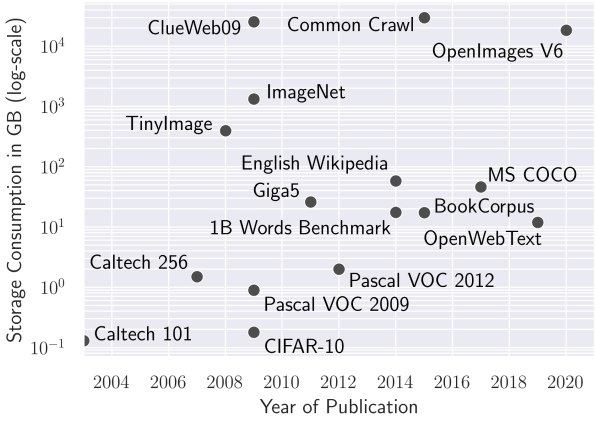


Figure 1: Storage consumption of real-world CV and NLP datasets over time on a logarithmic scale. CV: [16, 22–24, 29, 45, 47, 53, 83], NLP: [1, 11, 12, 14, 67, 92, 98].

The process to train a DL model consists of repeatedly iterating over the entire training dataset, measuring up to hundreds of iterations depending on the task at hand and the model complexity [34, 69, 72, 81]. Popular datasets show exponential storage consumption increase over time (Fig. 1), which makes data preprocessing harder, as local processing is not viable anymore due to memory limitations. Both distributed storage solutions [90] as well as distributed processing [5, 25, 55, 87, 95] can lead to new difficulties with network I/O and latency, which makes data preprocessing an integral part of the end-to-end DL pipeline. A Google study on their cluster fleet showed that the preprocessing pipeline takes more than a third of the total preprocessing time for 20% of their jobs [59].

Optimizing the model training is an active research topic that focuses on decreasing the total training time and increasing the data ingestion rate of the training process. There are many methods on training performance optimization and model optimization for both single GPU [30, 56, 85] and multi-GPU setups [36, 38, 74, 77] which allow for horizontal scaling with more hardware [63]. Recent hardware innovations help with improved model performance (cf. Fig. 3). Therefore, it is essential to optimize preprocessing pipelines to keep up with the training process speed.

The preprocessing part of a DL pipeline consists of multiple successive data transformation steps applied to the initial dataset until the final data representation matches the model input dimensions. This transformation can be performed once before the training or in every iteration while the training is happening. For example, CV pipelines from DL models that established new landmarks at their respective times [34, 49, 78, 81, 96] follow a common pattern of preprocessing steps: *read* the image from a storage device, *decode* it into an RGB matrix and *resize* the image to fit the model input dimensions. These steps can be followed by data augmentation, e.g., *pixel-centering*, *random-cropping* or a *rotation*, depending on the particular use-case. Preprocessing the full dataset once before training is viable if one wants to avoid the processing overhead in every iteration.

However, the final data representation and the storage device can negatively affect the preprocessing throughput. The training process's data ingestion can be throttled by I/O bottlenecks when loading the preprocessed data. The storage consumption typically increases at later preprocessing steps, as the corresponding data representations often store data inefficiently to facilitate processing (e.g., JPG [89] vs. an RBG matrix). This additional storage consumption can be a determining factor that slows down the final throughput. The file system, storage device, and the data loader from the DL framework may not be able to read the data fast enough (cf. Section 4).

We propose a new, more flexible way to look at DL preprocessing pipelines, where the decision for **each** preprocessing step to apply it *once* or in *every iteration* can be made freely based on quantifiable trade-offs. Such quantification can be provided by profiling.

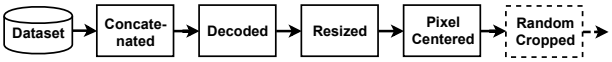


Figure 2: CV preprocessing pipeline

To motivate this new view on preprocessing pipelines, we performed experiments using different configurations of a CV pipeline (Fig. 2)[1]. Performing all preprocessing steps *at once* increases the throughput by 5.4× compared to *at every iteration* (Tab. 1). However, this increases storage consumption by more than 10×. In contrast, preprocessing the dataset *once* just until the resize step results in a 16.7× throughput increase while increasing the storage consumption only by 3.4× compared to processing all steps at every iteration.

Preprocessing strategy	Throughput in $\frac{samples}{s}$	Storage Consumption in GB
all steps at *every iteration*	107	**146**
all steps *once*	576	1535
until *resize* step, *once*	**1789**	494

Table 1: Trade-offs for the CV pipeline at different preprocessing strategies.

When comparing the data processing rate of a popular CV model, ResNet-50, to the different preprocessing strategies on state-of-the-art GPUs, we see that stalls on the A10, A30, and V100 can be prevented by using the optimal strategy (Fig. 3). In multi-node training setups and when using specialized hardware (TPUs), increasing preprocessing throughput demands becomes even more evident.

The idea of opening up the preprocessing pipeline and the resulting trade-offs have not been explored yet in a comprehensive fashion. This void prevents ML practitioners from optimizing their end-to-end DL pipelines. They lack guidance in how to do that, as well as tooling support to automate such optimizations.

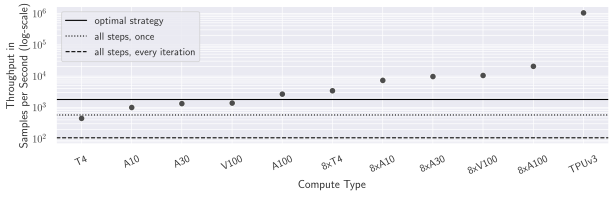


Figure 3: Throughput of ResNet-50 [34] for different hardware configurations. Black lines show the preprocessing throughput for the different strategies from Tab. 1. GPU profiling data from NVIDIA [63] and TPUv3 by Ying et al. [93].

In this paper, we close this research gap by performing a comprehensive analysis of preprocessing pipelines from a broad range of different ML domains. In doing so, we present practical insights into the pipelines themselves as well as the methodologies to analyze bottlenecks and an automated tool to perform profiling of arbitrary pipelines. This opens up a new dimension in end-to-end ML system optimizations, which was not considered in prior works that targeted the pipeline optimization with respect to the model accuracy [43, 58].

Our contributions are:

(1) **We profile seven different real-world pipelines** and define the trade-offs and characteristics that allow practitioners to improve existing pipelines by optimizing at the location with the greatest impact on the training throughput. This way, we could improve training throughput by up to 3-13× compared to fully preprocessing once.
(2) We provide lessons learned, where we **summarize the problems and unexpected findings** we encountered that can limit pipeline throughput. For example, we found that storage consumption can affect the throughput negatively in different ways. These insights can be used to clear up common misconceptions, and practitioners can be more aware of the impact the preprocessing pipeline has on the training performance.
(3) We present an open-source **profiling library** that automates the decision of which preprocessing strategy to pick based on a user-defined cost model.

Our paper is organized as follows. In Section 2, we introduce a general model and terminology of preprocessing pipelines. The experimental setup and the library design are explained in Section 3. We present our pipeline analysis and our findings in Section 4. Our derived insights are summarized in Section 5. Related work is reviewed in Section 6. Other approaches for pipeline optimizations are discussed in Section 7 and we conclude the paper in Section 8.

2 PREPROCESSING PIPELINES

We begin by describing the set of problems one faces when preparing a dataset for the training process. This includes determining the hardware requirements, the decision of *where* to preprocess the data, and *when* to preprocess, both of which decisions affect the training throughput in multiple ways.

[1]The only step which has to be applied every iteration is *random-crop*, as it is not deterministic (dotted line).

The preprocessing pipeline can be split into steps which are ran *once* ($S_1 - S_m$), called "offline" henceforth, and steps which are performed *every iteration* ($S_{m+1} - S_n$), called "online". The set of preprocessing steps depends on the dataset and the model input, but generally, any transformation is a step, like cropping an image or encoding a word. A preprocessing *strategy* is processing up to (and including) a step offline, and the remainder of the pipeline is executed online. Such a split is accomplished by inserting a *save* step which encodes and writes the data to disk (after S_m), and a *load* step that reads that data from disk (before S_{m+1}).

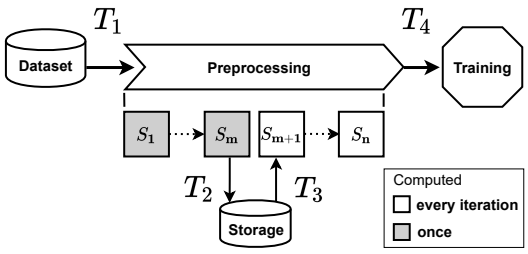


Figure 4: General DL preprocessing setup with different theoretical throughputs $T_1 - T_4$ between the preprocessing steps $S_1 - S_n$.

We conceptually divide the entire DL training process into three parts - the unprocessed dataset, the preprocessing pipeline, and the training (Fig. 4). They are all connected by four theoretical throughputs ($T_1 - T_4$), which can become processing bottlenecks.

T_1 is the read throughput from the dataset to the processing units which handle the *offline* preprocessing. This throughput is determined by hardware capabilities, such as storage devices, interconnects, processing capabilities, and software capabilities, such as file systems or DL frameworks.

T_2 is the write throughput from the offline computed preprocessing step(s) ($S_1 - S_m$) to a storage node. It is dependent on the throughput of each step, the interconnect to the storage node, and its write speed.

T_3 is the read throughput from the storage node to the processing units which handle the *online* preprocessing ($S_{m+1} - S_n$) and is subject to the same restrictions as T_1.

T_4 is the final preprocessing throughput when the data is ready to be fed into the training process. It is restricted by T_3, the online step(s) performance, and the interconnect to the training process. As T_4 limits the achievable training throughput, it is the most important to optimize.

In practice, it is often impossible to know the actual performance of future DL models or preprocessing pipelines. Only partial benchmarks are available to approximate the training throughput of popular DL models [63]. Even worse, there is no comprehensive throughput analysis of preprocessing pipelines, so one has to estimate the resulting T_4 throughput of a pipeline manually for every single deployment to prevent bottlenecks. Such an estimation is difficult to make, as there are many complexities involved.

One of those is the data encoding after step S_m, which serializes the entire dataset and places it on a storage device. The current default way to serialize datasets in two popular DL frameworks is the "pickle" encoding for PyTorch [68], and Protobuf [26] for TensorFlow [2]. Both encodings are not optimized for tensor data and may perform poorly. Applying an optimized compression algorithm may be useful but also introduces an additional online decompression step that may affect the T_4 throughput.

The deserialization throughput (T_1 and T_3) depends not only on the encoding but also on the storage solution and its interconnect to the nodes that run the preprocessing pipeline. A common storage solution in virtualized environments is Ceph [90], an object-based storage system. Such a complex and distributed system's performance depends on the storage hardware and the computing power and must be evaluated on a case-by-case basis. Without benchmarks for specific hardware setups, it is unclear how to split the pipeline to achieve the maximum throughput [58, 71].

Another variable to consider is the offline preprocessing time, as this may delay the training start. Long preprocessing times can be prohibitive if not amortized by faster training.

Additionally, some preprocessing steps that feature data augmentation (e.g., random-cropping for images) or shuffling the dataset have to be done online because their results are not deterministic and can not be cached for future epochs.

Some preprocessing steps decrease the dataset size and can make it fit in memory, which would be beneficial over multiple epochs to remove network read effects. However, preprocessing steps can also be computationally expensive and would better be processed offline which can increase throughput, even if they increase the dataset size. Both scenarios can benefit the throughput, but it is not obvious to determine without profiling whether caching a dataset or removing a CPU bottleneck is more effective at increasing throughput.

In conclusion, deciding how to split a preprocessing pipeline into offline and online steps is a complex problem. The importance of the trade-offs may depend on individual scenarios, such as preexisting hardware or framework dependencies, which can not be chosen freely. To solve this problem and provide guidance and best practices to ML engineers and users, a comprehensive analysis of common DL pipelines is needed that provides a structured overview of the pipelines' performance and insights about the individual steps' trade-offs. It is also necessary to evaluate whether profiling a small sample of the entire dataset is sufficient to estimate the total processing time, storage consumption, and T_4 throughput. This could reduce profiling overhead. Finally, a software solution should automate the profiling to quickly generate insights for a specific setup to optimize any given pipeline.

3 EXPERIMENTS

We analyze four different ML domains to showcase trade-offs in preprocessing pipeline optimizations: CV, NLP, Audio, and non-intrusive load monitoring (NILM). We evaluate the pipelines with in total seven different datasets in order to compare the impact of different encodings and image resolutions on the respective pipeline's performance. Every pipeline is based on common preprocessing steps from popular models and datasets in their respective domains. We assume that the training throughput is unbounded for our analysis, as we are interested in maximizing T_4 irrespective of the actual model. This section showcases the design of our profiling library, the individual pipelines and defines the experimental setup.

3.1 PRESTO Library

After initial manual profiling attempts, we decided to create the **Pre**processing **St**rategy **O**ptimizer (PRESTO) library that automates the generic pipeline profiling process. The library can be used with any preprocessing pipeline written with the TensorFlow `tf.data` API [59], and hence, is readily applicable to different use cases.

PRESTO contains a `Strategy` wrapper class that splits the preprocessing pipeline at any given step into an offline and online part. This is done by inserting a serialization and loading step at the *split position* with the TFRecord format, a wrapper around the Protobuf encoding [26] for TensorFlow. Additional parameters include the parallelism, sharding, caching behavior, and compression format, as well as the temporary logging directory.

The strategy wrapper class executes the entire preprocessing pipeline through the `tf.data` API. We simulate the training process by accessing the sample tensor's shape member to measure the preprocessing throughput without training an actual model. This allows us to profile the preprocessing pipeline's throughput by calling the `profile_strategy()` function. It accepts two parameters which have to be set manually: `sample_count` and `runs_total`. While it is useful to get an initial understanding of a pipeline's performance with less samples, some bottlenecks only show after local caches are full or a network link is used to its maximum capacity, so that we recommend profiling with the entire dataset.

Profiling focuses on three key metrics - *preprocessing time*, *storage consumption* and *throughput* - which can be easily tracked by internal Python code. For more in-depth information, `dstat` is executed in parallel and provides specific system-level information, like disk read/write loads and network traffic in case of network storage. These stats and additional metadata, like a unique hash and the split position, are returned as Pandas dataframes [91].

After the profiling is finished, the `StrategyAnalysis` class summarizes the findings and provides a semi-automatic way to pick the best strategy based on an objective function. The function normalizes the individual metrics to the range of [0,1] based on their minimum and maximum values and multiplies them by user-defined weights $w_{p,s,t}$. Let preprocessing time be $\mathbf{p}$, storage consumption $\mathbf{s}$, and throughput $\mathbf{t}$ as vectors of the respective values for all strategies:

$$f(w_p, w_s, w_t, \mathbf{p}, \mathbf{s}, \mathbf{t}) = w_p \times |\mathbf{p}| + w_s \times |\mathbf{s}| + w_t \times |\mathbf{t}|$$

The weights $w_{p,s,t}$ are can be defined manually, based on the user's objective. As an example, we want to find the optimal strategy to apply hyperparameter tuning on a model before a deadline. That means we want a low preprocessing time and the highest possible throughput, while the storage consumption is irrelevant. In this case, the weights would look as follows:

$$(w_p, w_s, w_t) = (1, 0, 1)$$

On the contrary, if we have access to a cluster with a lot of compute power and are not in a race against time, it will be preferable to sort only by throughput ($w_{p,s} = 0, w_t = 1$), which is a good default configuration. This procedure can be applied to every strategy, which can have different parallelization, sharding and compression options and lead to new trade-offs. More complex objective functions can feature cloud providers' processing and storage prices. We presume that renting a low-cost VM and profiling the different strategies could probe the infrastructure, i.e., network bandwidths. This allows us to extrapolate the processing performance by tensor-specific CPU benchmarks like PASTA [52] for high-cost VMs. We provide our library as an open-source project at https://github.com/cirquit/presto.

3.2 Pipelines

We profiled seven pipelines from four different domains and designed them to represent popular DL models. Table 2 shows the seven datasets we used to profile the pipelines with their storage consumption, sample count, and format. The datasets and pipelines show a variety of common formats, and different intermediate sizes, e.g., the NLP pipeline has a strategy that increases the initial storage consumption by 64×, while NILM has a strategy that decreases the initial storage consumption by a factor of 12× (Fig. 6). The pipelines were implemented with the `tf.data` API [59] which automates pipeline execution and allows us to parallelize computations easily. We define a *sample* in this context as data that is used as input for a DL model.

The naming of the steps in the pipelines follows a common pattern. First, the data is read from disk (`unprocessed`). After reading the dataset from disk, a concatenation step transforms the input files into a single TFRecord binary in order to allow for efficient sequential read access (`concatenated`). The concatenation step was technically not feasible for the Audio pipeline, and was omitted for the NILM pipeline as the raw data was already stored in concatenated binary form. Then, the data is decoded into a tensor format (`decoded`). Finally, additional transformation steps can be applied to bring the data into a format suitable for the training process. Generally, the steps have two characteristics: the online processing time and the relative increase or decrease of storage consumption. We explain the trade-off between these two characteristics in Sec. 4.1, which can change for the same step just by using different datasets, e.g., decoding can increase or decrease the storage consumption depending on the initial file encoding (e.g., JPG vs. PNG).

The performance of preprocessing steps depends not only on the implementation of the step itself, but also on its position in the pipeline and on the input data. We specifically showcase this behaviour in Sec. 4.6 by changing the position of a new step in an existing profiled pipeline.

Dataset	Pipeline	Sample Count	Size in GB	Avg. Sample Size in MB	Format
ILSVRC2012 [75]	CV	1.3M	146.90	0.1147	JPG
Cube++ JPG [19]	CV2-JPG	4890	2.54	0.5203	JPG
Cube++ PNG [19]	CV2-PNG	4890	85.17	17.4176	PNG
OpenWebText [1]	NLP	181K	7.71	0.0427	TXT
CREAM [40]	NILM	268K	39.56	0.1477	HDF5
Commonvoice (en) [6]	MP3	13K	0.25	0.0197	MP3
Librispeech [66]	FLAC	29K	6.61	0.2319	FLAC

Table 2: Metadata of all profiled datasets.

3.2.1 CV. We profile three datasets with the CV pipeline (Fig. 2) to analyze the performance under different image resolutions and encodings. ILSVRC2012 [75] is a low resolution, JPG encoded subset of ImageNet [16] and is a popular and commonly acknowledged dataset for visual object recognition. Cube++ is a high-resolution dataset and comes in two flavors: as 16-bit encoded PNGs and JPGs [19]. The difference in storage consumption between the two encodings allows for a direct comparison of the decoding performance. The images from Cube++ are roughly 5× larger than in ILSVRC2012, which allows to analyze how much the image resolution affects the throughput of each strategy. Details of this pipeline have been discussed in Section 1.

3.2.2 NLP. The NLP pipeline (Fig. 5a) is based on GPT-2 [73], a popular transformer-based model which tries to predict the next word based on the textual input. We used the dataset from the corresponding open-source implementation [1] of OpenWebText, which was replicated from the GPT-2 paper. Our version of the dataset is an early iteration and takes up 8 GB compared to the most current one at 12 GB. The dataset consists of HTML content from scraped URLs that have been upvoted on Reddit, a social media platform, as an indicator of human interest and intelligible content. It is stored as multiple text files.

The preprocessing starts with reading text files (`concatenated`) and decoding the actual textual content (`decoded`) with the same HTML parsing library (newspaper [64]) as GPT-2. Each word is encoded into an `int32` via Byte Pair Encoding [76] (`bpe-encoded`), which is then looked up in a word2vec embedding [57] that returns a `float32` tensor of dimension 1×768 (`embedded`). This vector is stacked for every word in the text, resulting in an $n \times 768$ tensor, the final model input. These preprocessing steps' complexity depends on the tokenization model and the final embedding, making their performance hard to predict.

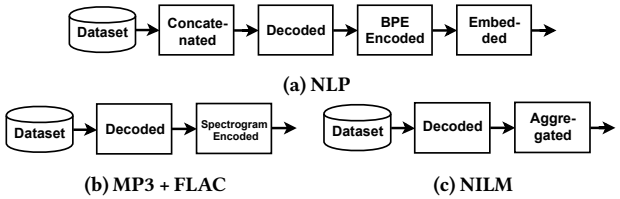


Figure 5: Preprocessing pipelines

3.2.3 Audio Processing. For the audio pipelines, we took inspiration from Baidu's Deep Speech model [3, 31]. Deep Speech is an RNN-based model that translates spoken audio samples to text. Both preprocessing pipelines (Fig. 5b) decode the compressed audio signal into the raw waveform (`decoded`) of the size $l \times r$, where l is the sample duration in seconds and r is the sampling rate encoded as `int16`. The waveform is transformed using a short-time Fourier transform (STFT) with a window size of 20 ms and a stride of 10 ms. The spectrogram is then transformed using an 80-bin mel-scale filter bank, leading to a size $\frac{l-20ms+10ms}{10ms} \times 80$ tensor encoded as `float32` (`spectrogram-encoded`). The difference between the pipelines is their respective input format (MP3 vs. FLAC). In contrast to some implementations [31], we do not convert the data to mel-frequency cepstral coefficients (MFCCs) because it has been found that DL models work as well or better without this transformation [35, 70, 79].

As datasets, we use the Mozilla Commonvoice 5.1 English corpus [6] for MP3 files and the Librispeech dataset [66] for FLAC files.

3.2.4 NILM. Our signal processing pipeline is based on MEED [39], a state-of-the-art event detection model used for non-intrusive load monitoring of electrical data. The task is to classify individual appliances based on the aggregated voltage and current reading measured on a building's mains. These datasets typically have a very high frequency, e.g., 6,400-50,000 Hz [4, 40, 46] to provide information on subtle changes that can be useful for appliance identification.

We used CREAM [40], a component-level electrical measurement dataset for two industrial-grade coffeemakers encoded as HDF5 files per hour. CREAM contains two datasets from two different coffee machines (X8 and X9), from which we used the larger X8 dataset because it takes up more than double the storage consumption of X9, i.e., totals 744 hours of 6.4 kHz sampled current and voltage. This dataset's fundamental difference to the other datasets is the `float64` encoding, which is favorable for NILM tasks [41] but introduces additional storage consumption.

The pipeline starts by reading HDF5 files and extracts the voltage and current signals from them (`decoded`). They are sliced in 10-second windows, which results in a 2×64.000 tensor of `float64`. Then, three aggregated values are computed: the reactive power [8], the root-mean-square of the current, and its cumulative sum [39, 84, 99] (`aggregated`). These aggregation operators work with a dataset period length of 128, which results in a tensor of size 3×500 encoded as `float64`.

3.3 Experimental Setup

We execute our experiments on a virtual machine with 80 GB DDR4 RAM, 8 VCPUs on an Intel Xeon E5-2630 v3 8x@2.4 GHz with an Ubuntu 18.04 image on our OpenStack cluster. Our Ceph cluster, backed by HDDs, is used as a storage device via *cephfs*, with a 10 Gb/s uplink and downlink. This storage is used for both storing the intermediate dataset representations as well as the unprocessed datasets. We repeat each experiment five times and we drop the page cache after every run to remove memory caching effects except for explicitly marked caching experiments. All experiments are run with 8 threads except for explicitly marked scalability experiments with a sharded dataset so that every thread has an assigned individual file to read in parallel. All experiments are executed with Python 3.7 and TensorFlow 2.4. Specific library versions are available in our GitHub repository.

4 ANALYSIS

By profiling the preprocessing pipelines, we aim to provide insights about how to pick the optimal preprocessing strategy for a specific set of hardware, the datasets, and the characteristics of each pipeline. The goal is to maximize the T_4 throughput (Fig. 4) while keeping the storage consumption and offline preprocessing time low. Our analysis is focused on four core aspects:

Storage Consumption versus Throughput: A high storage consumption can render extensive offline preprocessing useless or even counterproductive to achieve high throughput. This has two causes. First, the dataset may be split into many files, and the storage does not respond fast enough to random read access, leading to a storage bottleneck. Secondly, specific preprocessing steps (e.g., normalizing an integer range to floating points) inflate the data volume. This can lead to a lower throughput because the saved preprocessing time is outweighed by the increased data ingestion time (see Section 4.1).

Caching: As a DL training process typically iterates over the dataset multiple times, there can be an increased throughput due to caching effects after the first epoch. However, this effect depends on whether the preprocessed training data fits into memory. Further, reading the cached dataset from memory can help to isolate processing bottlenecks from storage bottlenecks (see Section 4.2).

Compression: Compression provides a way to trade off CPU overhead for en-/decode steps against smaller storage consumption. We profile each strategy with GZIP [17] and ZLIB [18] compression and show under which circumstances compression improves the throughput (see Section 4.3).

Parallelization Capabilities: Preprocessing steps can hinder performance if multi-core CPUs are not utilized effectively. Scalability bottlenecks may have a substantial impact on throughput. We compare the speedup of each preprocessing step under multi-threading and system-level caching in Section 4.4.

We also touch on shuffling (see Section 4.5) and discuss how to introduce new steps to an already profiled CV pipeline based on a case study in Section 4.6.

Threads	Files per Thread	Bandwidth	Latency	IOPS
1	1	219 MB/s	4–10μs	53400
8	1	**910 MB/s**	4–10μs	**222000**
1	5000	6.6 MB/s	4–10μs	1629
8	5000	40.4 MB/s	4–10μs	9853

Table 3: `fio` profile of our storage cluster

4.1 Storage Consumption versus Throughput

We profiled the throughput and storage consumption for all strategies of each pipeline in Figure 6. The theoretical network read speed to the HDD-backed Ceph storage cluster is capped at 1.25 GB/s (10 Gb/s) due to hardware limitations, but the actual rates differ based on the access pattern. To provide a pipeline-independent measurement, we profiled four workloads with the `fio` tool [21] to simulate both sequential and random file access with 5000 files of 0.2 MB each and with one 5 GB file, which is comparable to our `unprocessed` and `concatenated` strategies. Additionally, we tested the single-threaded performance compared to 8 threads with the same workload per thread. Table 3 shows that reading sequentially is 33× faster for single- and 22× faster for multi-threaded execution. While our single-threaded bandwidth is limited to 219 MB/s, the multi-threaded execution is close to the hardware cap with 910 MB/s. This helps to explain our main observations:

Pipeline	Throughput in SPS		Network Reads in MB/s	
	`unprocessed`	`concatenated`	`unprocessed`	`concatenated`
CV	107	**962**	(±3) 12	(±16) **111**
CV (SSD)	588	**944**	(±8) 68	(±15) **108**
CV2-JPG	88	**288**	(±11) 46	(±72) **110**
CV2-PNG	15	**21**	(±37) 270	(±54) **390**
NLP	6	6	(±0.2) 0.21	(±1.5) **0.26**
NLP (SSD)	3	3	(±0.2) **0.17**	(±1.2) 0.16

Table 4: Throughput and average network read speeds for strategies with concatenation.

(1) Concatenating can increase throughput significantly.

An I/O bottleneck may arise when the storage cannot saturate the hardware bandwidth based on the data access pattern. Out of the four pipelines which have a `concatenated` strategy (CV, CV2-JPG, CV2-PNG, and NLP), we see that all CV-based pipelines have a throughput increase between 1.4× and 9× compared to the `unprocessed` strategy (Table 4). The individual differences in the throughput can be explained by the dataset size and the average storage consumption of a sample (Table 2). Due to the CV samples being smaller than the CV2-JPG samples, CV achieves 962 SPS compared to 288 SPS while having a similar network read speed of approximately 110 MB/s. The CV2-PNG dataset has a sample storage consumption of 17.4 MB and the network read speed increases from 270 MB/s to 390 MB/s with concatenation.

Contrary to CV-based pipelines, the NLP pipeline does not benefit from concatenating as the throughput stays at 6 SPS, indicating a CPU bottleneck. This bottleneck is resolved in the `decoded` strategy, where throughput increases significantly (Fig. 6d).

As an HDD-based storage solution is particularly vulnerable to random access bottlenecks, we additionally profiled the performance of the CV and NLP pipeline on an SSD-backed Ceph cluster. The CV `unprocessed` strategy had a throughput of 588 SPS, which is almost 6× faster than the HDD experiments. However, at the `concatenated` strategy, both HDD and SSD reach roughly the same throughput (962 SPS vs. 944 SPS), i.e., at sequential access the SSD-backed storage is not faster. For NLP, the SSD storage did not provide a better throughput as it still faces the CPU bottleneck at the `concatenated` strategy.

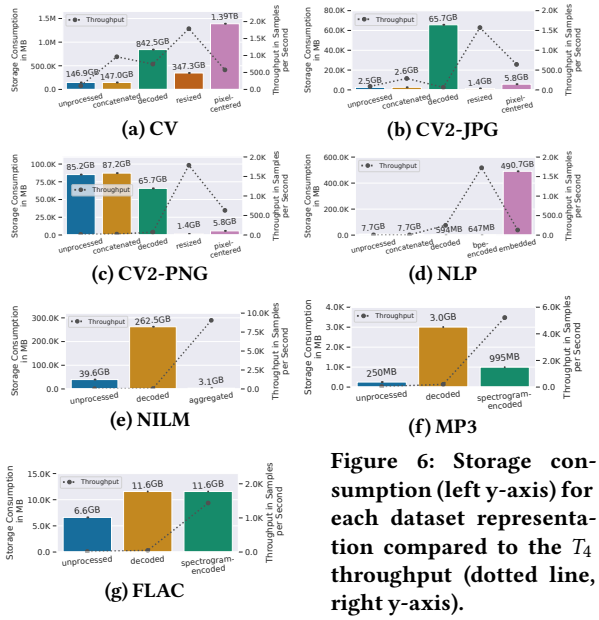


Figure 6: Storage consumption (left y-axis) for each dataset representation compared to the T_4 throughput (dotted line, right y-axis).

(2) The maximum throughput of a strategy is influenced by its storage consumption.

When the CPU performance in combination with a storage setup can saturate the hardware bandwidth (i.e., in our case, read data with 1.25 GB/s), then a maximum theoretical throughput can be calculated by dividing the *bandwidth* by the *storage consumption per sample*. This theoretical throughput is based on two steps. First, one reads a sample from the storage into memory. Second, one applies the online transformation steps in succession until the sample can be fed into the training process. The total time of these two steps defines the pipeline's throughput (i.e., samples per second), hence also the network read speed (i.e., MB per second). In our case, the actual throughput we can achieve is bound by the multi-threaded read performance to our cluster, which is at 910MB/s with eight threads (Table 3). This profiled network read speed provides a baseline of the maximum possible throughput. The goal of every strategy should be

to have a short enough transformation step to achieve this baseline read speed. In turn, if transformation steps are too long, such that the maximum read cannot be reached, we can assume a CPU bottleneck.

A characteristic result of how storage consumption affects the throughput of strategies can be seen in the CV, CV2-JPG, CV2-PNG and NLP pipelines (Fig. 6a, 6b, 6c, 6d), where the last strategy has the least amount of online processing to do, but performs worse than its corresponding preceding strategy. An excellent example of this is the CV pipeline. At the last strategy, `pixel-centered`, we have an average network read speed of 585 MB/s and need to read 1.4 TB of data. This stands in contrast to the previous strategy, `resized`, which has a lower network read speed of 470 MB/s, but only needs to read 347 GB. Therefore, the `resized` strategy has a more than 3× greater throughput of 1789 SPS compared to `pixel-centered` (576 SPS), even though `resized` applies more processing steps online. The cause of the increased storage consumption is that `pixel-centered` converts each pixel from an `uint8` to a `float32` which effectively quadruples the storage consumption. All our CV-based pipelines share this characteristic at different magnitudes, which results in the `resized` strategy having the best throughput.

The NLP pipeline has a similar issue with the embedded step, which slows down the throughput from 1726 SPS with `bpe-encoded` to 131 SPS with `embedded` (a factor of 13×). Applying the embedding step online is very computationally intensive, which yields a data ingestion of only 6 MB/s for `bpe-encoded`. One could think that preprocessing this step offline should improve performance. But this is not the case, because the storage consumption increases from 647 MB to 491 GB, such that the benefit of processing the embedding step offline is outweighed by the increased time to read the dataset.

The remaining pipelines, NILM, MP3 and FLAC (Fig. 6e, 6f, 6g), share the common characteristic that the last preprocessing step is the most computationally expensive one, which leads to the best throughput when processed offline. While they all have different storage consumption, none of the pipelines approaches the maximum possible network read speeds at their respective last strategy.

On first sight, this is counter intuitive. In the last strategy, there is almost no processing to be done except for decoding the read data. Why do these strategies not approach the network read limit? A deeper investigation leads to our following observation.

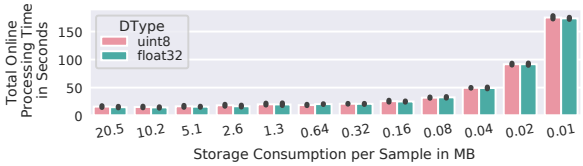


Figure 7: Profiling a synthetic 15 GB dataset with different datatypes and sample sizes.

(3) A high storage consumption per sample allows for easier I/O bandwidth saturation.

A deserialization step is applied onto every sample which is read from the storage to transform it from a bytestream into a tensor. We observed that an increasing sample size positively influences the I/O bandwidth of reading and deserializing. To provide a basis to our observation, we conduct an experiment with synthetic data that shows the effect of different sample sizes on the online processing time of reading and deserializing. We evaluate sample sizes from 0.01 MB to 20.5 MB with doubling increments while keeping a total storage consumption of 15 GB for both `uint8` and `float32` datatypes which are common in our real-world pipelines. To keep the same total storage consumption with different sample sizes we adapted the sample count, which ranged from 732 (20.5 MB) to 1.5M (0.01 MB) samples. Figure 7 shows the result that reading the same amount of data with different sample sizes has a major effect on the processing time. A dataset consisting of large (20.5 MB) samples takes less than half the processing time of small ($\leq$ 0.08 MB) samples. At a sample size of 0.01 MB, it takes more than 11× longer to process the 15 GB of data compared to 20.5 MB samples. Finally, the different data types do not have an impact on the processing time, as both `uint8` and `float32` show similar results.

A good example for this observation is the comparison of the `decoded` strategy between CV (Fig. 6a) and CV2-JPG (Fig. 6b). The average sample size is 13 MB for the `decoded` strategy of CV2-JPG with a network read speed of 828 MB/s, which indicates an I/O bottleneck. However, with the same strategy, the CV pipeline has a sample size of 0.6 MB and the average network read rate is at 491 MB/s, which is not even close to our maximum bandwidth. Notably, the CV pipeline has a lower computational load compared to CV2-JPG *and* has to read fewer data from storage with each sample due to the small sample size. But it still does not manage to saturate the I/O bandwidth. Therefore, the CV `decoded` strategy suffers from a CPU bottleneck. To further validate our assumption, we profiled the CV pipeline with 16 threads which increased the network read speed by 64 MB/s and improved the throughput by 142 SPS. The additional multi-threading also increased the throughput by 583 SPS and 100 SPS for the `resized` and `pixel-centered`, respectively. All last strategies like `aggregated` (Fig. 6e), `spectrogram-encoded` (Fig. 6f, 6g), and `embedded` (Fig. 6d) share that characteristic and do not saturate the I/O bandwidth (96 MB/s, 317 MB/s, 564 MB/s and 315 MB/s respectively).

4.2 Caching

DL training jobs typically run over multiple epochs, which means the dataset is read multiple times and could benefit from being cached in memory after the first epoch. We evaluated the throughput of all pipelines over two epochs for all strategies. In this set of experiments, we do not flush the page cache after the first epoch. Our observations are as follows:

(1) Caching is not beneficial when the storage consumption is higher than the available memory.

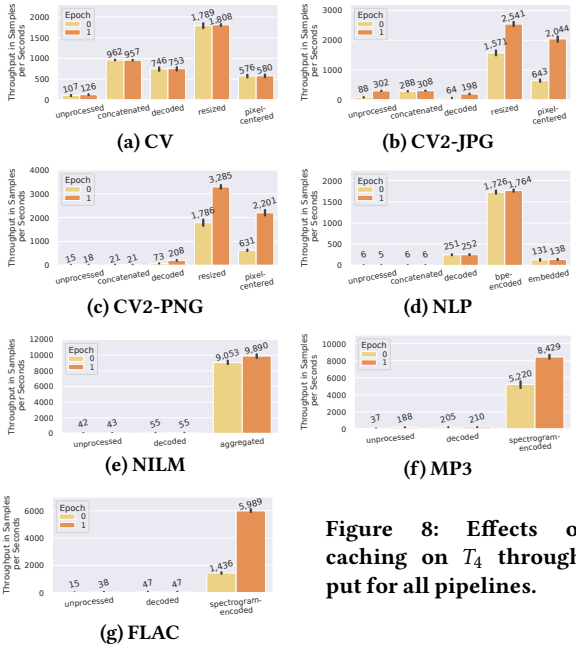
If the data set is too large for the memory, the dataset is read completely from the storage at every epoch. Hence, throughput is not increased by caching. All strategies that have a storage consumption higher than 80 GB (Fig. 6) have the same throughput over all epochs (Fig. 8).

(2) Caching does not remove CPU bottlenecks.

Assuming that the dataset fits into memory, caching can only improve throughput significantly if there is no CPU bottleneck. Reading data from memory is much faster than from remote network storage, but the impact of fast data access can become insignificant when followed by computationally expensive preprocessing steps. An excellent example of this is the NLP pipeline (Fig. 8d). The first two strategies `unprocessed` and `concatenated` have the same throughput of 6 SPS over all epochs because decoding is very compute intensive while the datasize is relatively small (7.7 GB). Then, after

decoding the data (594 MB), we face a new computationally expensive step, byte-pair encoding, which transforms the text into integers and increases the storage consumption to 647 MB. This is followed by the embedding step, which also takes much time. All of these strategies face a CPU bottleneck while having a storage consumption that allows the data to be cached. Finally, at the `embedded` strategy, the dataset only needs to be read from the storage, but grows in size to 490.7 GB, such that caching has no impact on throughput.

Similar CPU bottlenecks can also be observed in the `unprocessed` strategies of CV2-PNG, NILM, MP3 and FLAC (Fig. 8c, 8e, 8f, 8g), the `concatenated` strategies of CV2-{JPG,PNG} (Fig. 8b, 8c) and the `decoded` strategies of MP3 and FLAC. The remaining strategies (`resized`, `pixel-centered`, `aggregated`, `spectrogram-encoded`) benefit from caching the most as they have low storage consumption and are not followed by computationally expensive steps. However, caching improves the throughput with differing factors (1.1×-4.2×), which results in the next observation.



(a) CV (b) CV2-JPG (c) CV2-PNG (d) NLP (e) NILM (f) MP3 (g) FLAC

Figure 8: Effects of caching on T_4 throughput for all pipelines.

(3) System-level caching performance is affected by the storage consumption per sample.

While cached data removes the performance impact of remote storage, the preprocessed dataset still has to be fetched from memory and deserialized. We confirmed this by comparing the trace log between epochs. To examine the memory bandwidth, we used `sysbench` [20] to profile our memory which resulted in 166 GB/s. This should theoretically yield a multiplicative increase in throughput, which we do not achieve because we are not close to the maximum I/O bandwidth (cf. Sec. 4.1 observation **(3)**).

We investigate this observation by profiling synthetic `float32` datasets with different sample sizes with both system- and application-level caching (Fig. 9). At the lower end of storage consumption, starting with 0.16 MB per sample, reading smaller samples takes increasingly longer. The smaller the sample size, the more processing time the deserialization takes, which lessens the final throughput of

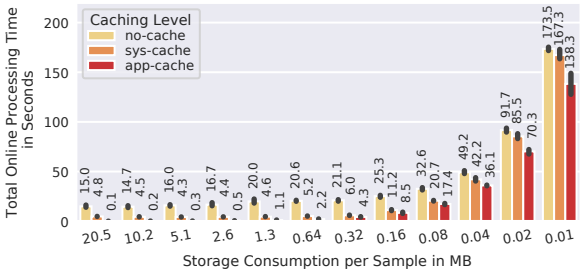


Figure 9: Online processing time for different caching levels and sample sizes of a synthetic 15 GB dataset.

the cached dataset. At 0.04 MB and lower, the processing time when data is cached in memory (`sys-cache`) is comparable to the case where data is on storage (`no-cache`), nullifying the effects of caching.

Our real-world experiments show the same behaviour. The storage consumption per sample for MP3 is 0.08 MB at the `spectrogram-encoded` strategy, while having a relative throughput increase of 1.6× (Fig. 8f) with caching. Meanwhile, the FLAC pipeline has a storage consumption of 0.4 MB per sample with the same strategy and increases its throughput by 4.2× (Fig. 8g) with caching. The NILM pipeline shows almost no increase in throughput (1.1×) over multiple epochs (Fig. 8e) as the sample size is only 0.012 MB.

While this is interesting, system-level caching via the page cache is somewhat unsatisfactory, since one wants to cache the tensor data and not be bottlenecked by the deserialization. The TensorFlow function `tf.data.Dataset.cache` caches the deserialized tensors in memory, avoiding deserialization overheads. This leads us to our fourth observation.

Pipeline	System-level	Application-level	Sample Size
CV2-JPG	3.3×	15.2×	1.18 MB
CV2-PNG	3.5×	14.5×	1.18 MB
FLAC	4.2×	8.0×	0.41 MB
MP3	1.6×	2.2×	0.08 MB
NILM	1.1×	1.4×	0.01 MB

Table 5: Throughput increase for different caching level compared to no caching of of each pipeline's last strategy.

(4) Application-level caching is more efficient than system-level caching, but is still affected by the storage consumption per sample.

To understand how application-level caching affects the performance, we profiled all pipeline's respective last strategies, as well as our synthetic 15 GB datasets again with application-level caching. The results of the synthetic datasets in Fig. 9 (`app-cache`) show that application-level caching is faster, but there is the same pattern of increasing processing time with a smaller sample size. The online processing time with application-level caching consists solely of reading the samples from memory. We can calculate the time spent on deserialization by subtracting the `app-cache` time from `sys-cache` time. By dividing with the `sys-cache` time, we can get the percentage of the time spent on deserialization. For the sample sizes 20.5 MB to 5.1 MB we spend 94-98% time on deserialization ($\frac{4.8-0.1}{4.8}$), compared to 14-18% for 0.08 MB to 0.01 MB ($\frac{167.3-138.3}{167.3}$). Hence, the largest relative gains with application-level caching can be achieved with large sample sizes.

Our real-world pipelines have shown a similar throughput improvement compared to system-level caching when the dataset fits into memory (Tab. 5). The decline in throughput improvement with both caching levels is directly correlated with a smaller sample size. The last strategies of the CV and NLP pipelines failed to run with application-level caching as the dataset did not fit into the cache (cf. Sec. 4.2 **(1)**).

4.3 Compression

Storage consumption has shown to be an important factor for throughput. Compression adds a new possibility to decrease storage consumption at the cost of an offline compression step and an online decompression step. For compression to provide a benefit, the gains of decreased data size must outweigh the computational overheads. A common metric to evaluate compression on storage consumption is the *space saving* percentage. For example, if the size did not change after compression, the space saving is 0%. When it changes from originally 5 GB to 1 GB, the space saving is 80%. We omitted the `unprocessed` strategy for all pipelines because accessing single files is bound by the random access performance of the storage (cf. Sec. 4.1 **(1)**) and compression does not help with this issue. The results of our compression experiments are shown in Fig. 10. We make the following observations:

(1) High space savings do not guarantee improved throughput.

Space saving affects the throughput positively for some, but not all strategies. All CV-based pipelines had an increase in throughput with compression between 1.6× and 2.4× at `pixel-centered` where space saving is between 73% and 93% (Fig. 10a, 10c, 10e). In this case, the faster read time in total was beneficial compared to the cost of the additional decompression step.

In contrast, the strategies of the NLP pipeline have a space saving between 28% and 80%, but none of them had a throughput increase (Fig. 10g). The reason for that is that every strategy was bound by a computationally expensive CPU step except the last strategy `embedded`. At `embedded`, the dataset is only read from disk and deserialized, but the space saving of 28% was not enough to benefit the total throughput.

The same effect becomes visible when comparing the strategies `decoded` and `resized` between the CV2-PNG and CV2-JPG pipelines (Fig. 10c, 10e). The only difference between the pipelines is the encoding of the images, with JPG being a lossy storage format, while PNG is lossless. The CV2-PNG pipeline has a better space saving with compression with the `decoded` strategy (83%) compared to CV2-JPG (41%). This results in a throughput increase by 1.5× for CV2-PNG compared to the 89% throughput deterioration at CV2-JPG. The `resized` strategy with the PNG images has a space saving of 81% and improves the throughput by 1.3×, while JPG only saves 24% of space and reduces the throughput to 96%. The compression artifacts introduced by the lossy JPG encoding affect the space saving of both GZIP and ZLIB negatively.

[1]Unfortunately, after one full run with each compression library (GZIP, ZLIB) for each strategy of the CV pipeline, our CEPH storage system was reconfigured which led to non-comparable results of the respective repeat runs. Hence, for these specific experiments, we only report results of one run (instead of the average of five runs, as in all other experiments).

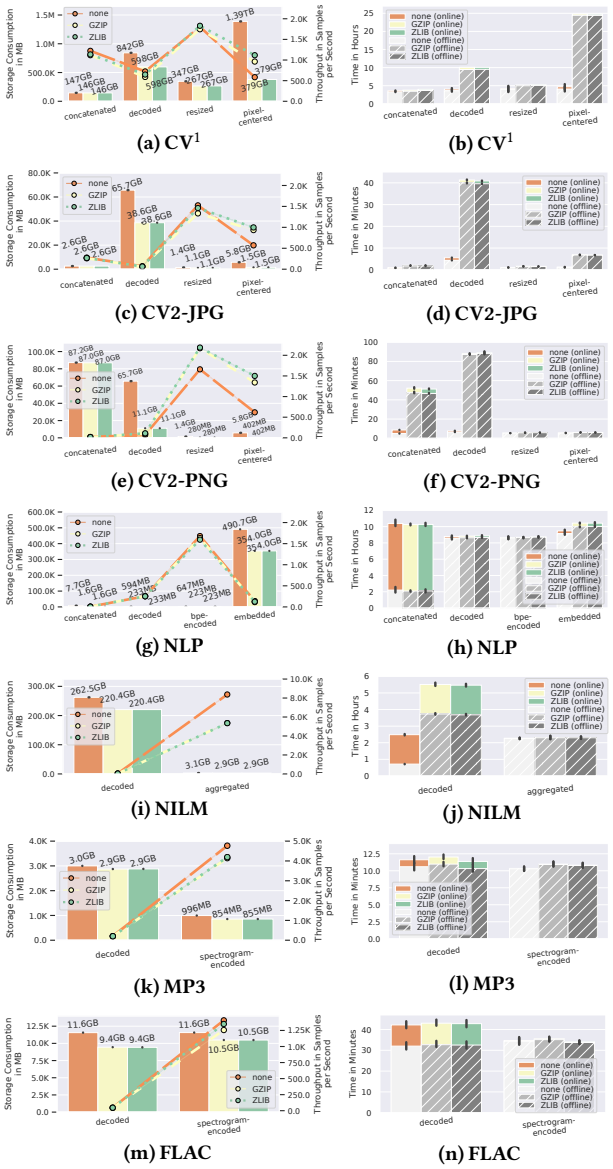


Figure 10: Left column: Storage consumption compared to T_4 throughput (dotted lines) with compression. Right column: Offline (grey hatched bars) and online processing time (colored) with compression.

All the other pipelines, NILM, MP3 and FLAC, slow down with compression and have a varying space saving between 0.3-41.2% (Fig. 10i, 10k, 10m). When comparing the different compression types, ZLIB was slightly faster and had a comparable space saving to GZIP, except for NLP's `bpe-encoded`, where it was slightly slower compared to GZIP.

(2) Offline compression and write time can be volatile.

When compressing a dataset, the processing time is increased by the compression algorithm, and decreased by the lower write time due to lower storage consumption. The balance between these

steps is not predictable from our observations, as the CV2-PNG pipeline shows (Fig. 10f). With the `concatenated` strategy and a space saving of only 0.3%, it takes 9.6× longer to save the dataset to storage. The strategy `decoded` has a space saving of 83% and takes 13.5× longer for offline processing. The next strategies, `resized` and `pixel-centered`, have a space saving of 80%-93% and only take 1.08-1.1× longer. Compared to a slightly worse space saving of 74% with the `pixel-centered` strategy at the CV2-JPG pipeline (Fig. 10d), the offline processing time is increased by 6.1×.

Generally, we see examples of a high space saving and no effective increase in offline processing time in NLP (Fig. 10h), a low space saving with a higher offline processing time in NILM (Fig. 10j) and CV2-PNG `concatenated` (Fig. 10f), and low space saving with no effective increase in offline processing time in MP3 (Fig. 10l), FLAC (Fig. 10n), and CV `concatenated` and `resized` (Fig. 10b). Space saving does not seem to be a good predictor at how the compression will affect the offline processing time.

4.4 Parallelization Capabilities

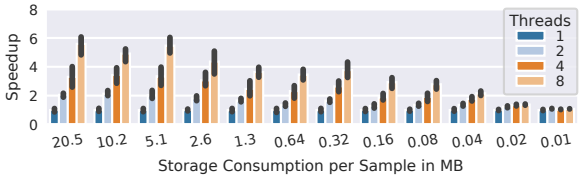


Figure 11: Reading a synthetic 15 GB dataset with different sample sizes to compare multi-threaded scalability.

We analyzed the parallelization capabilities of each pipeline under multi-threading, as this is one of the best practice recommendations to speed up data pipelines and remove I/O bottlenecks [27]. We compared the speedup of each strategy by running it with 1, 2, 4, and 8 threads over two epochs with system-level caching enabled (Fig. 12). The profiling was done with a fraction of the dataset (up to 8000 samples) so that the second epoch could be fully cached for each pipeline to compare the speedup with system-level caching. We make the following observations:

(1) A small storage consumption per sample hinders multi-threaded performance. We know from the previous Sections 4.1 and 4.2 that a small storage consumption per sample affects the online preprocessing time negatively. However, how does it affect multi-threaded execution? To analyze this, we reused the same synthetic 15 GB `float32` dataset with different sample sizes to compare their multi-threaded read and deserialization time. The results in Fig. 11 show a similar trend as before, with a speedup of close to 1× for the 0.01 MB sample sizes. This means that processing small samples with a single thread takes equally long as with eight threads. We traced the issue down to an increased amount of context switches with smaller sample sizes (100,000 per second at 0.01 MB compared to 5,000 per second for 20.5 MB). Additionaly, as we extracted from the trace log, every thread only processes a single sample at a time and is finished faster with smaller sample sizes before being scheduled again. Scheduling a thread to process a new sample induces so much overhead that multi-threading can not be effective at small sample sizes.

A good example from our real-world pipelines is NILM (Fig. 12i) at the `aggregated` strategy, which has no effective speedup due

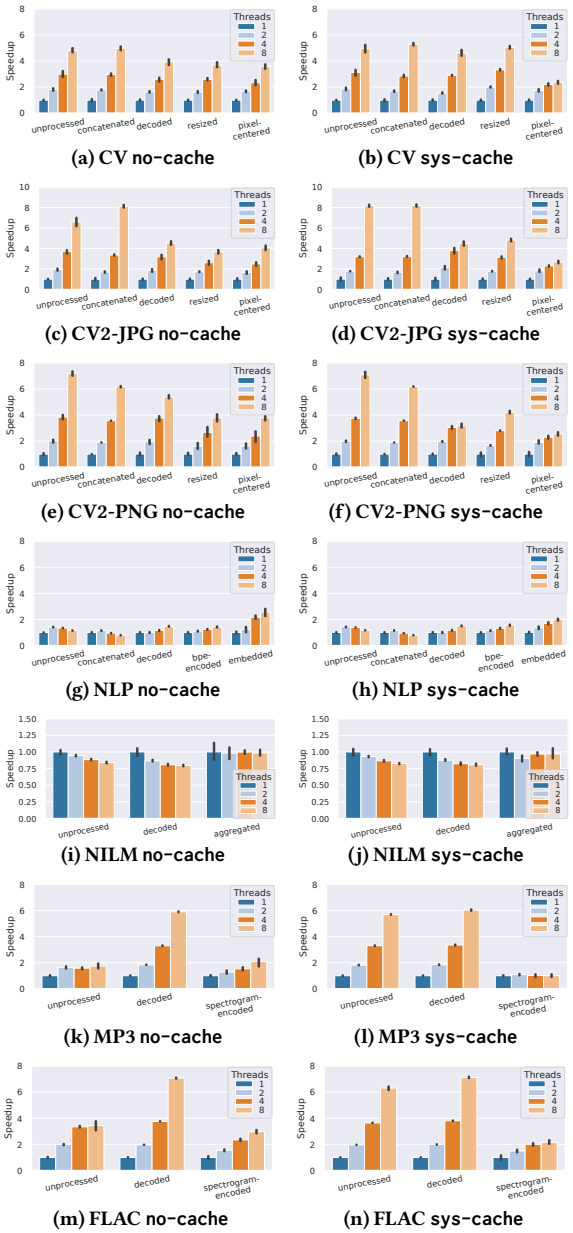
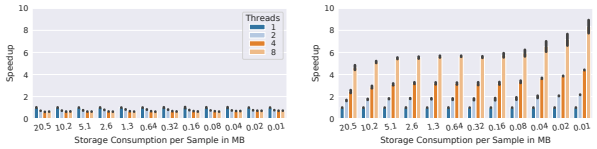


Figure 12: Speedup at 8000 samples. Left column: No caching. Right column: System-level caching.

to a sample size of 0.01 MB. Even when reading the dataset from memory (Fig. 12j), there is virtually no change in speedup. Every last strategy from each pipeline has as slightly worse speedup when reading from memory (`sys-cache`) than when reading from storage (`no-cache`) (Fig. 12). The reason for this is that memory provides a higher bandwidth, so reading data is fast, even with a single thread. Therefore, the effect of context switches is highlighted even more compared to the case where the slower network read speeds affects the total processing time additionally.

(2) Inefficient preprocessing can reduce multi-threading scalability.

While most of the scaling issues like `bpe-encoding` in NLP can be explained with a small sample size (0.003 MB), we also observed slowdowns (speedup < 1.0), which have a different root cause. This happens with the first two strategies of NILM (Fig. 12i, 0.15 MB and 0.98 MB) and NLP (Fig. 12g, 0.04MB), which are not alleviated by reading from memory (Fig. 12j, 12h, respectively). This points to a processing issue. One thing that both `unprocessed`, `concatenated` (NLP) and `decoded` (NILM) have in common, is that they are using external Python libraries like NumPy and newspaper wrapped in a `tf.py_function`, while all the other preprocessing steps are provided by the TensorFlow library.



(a) NumPy RMS (b) TensorFlow RMS

Figure 13: Speedup of applying RMS to a synthetic 15 GB dataset with different sample sizes implemented in NumPy and TensorFlow.

To test how external libraries affect the throughput, we created a new preprocessing step which applies the root-mean-square (RMS) function with a period of 500 over the entire sample, leaning on a similar computation from the NILM pipeline. We implemented this step both with NumPy and TensorFlow, and we profile our synthetic datasets with steps applied online individually. The results in Fig.13 show that the NumPy implementation has the same slowdown as NILM and NLP for all sample sizes, whereas the TensorFlow implementation shows a speedup between 4-8× for eight threads. However, while the NumPy implementation does not scale, it is still 2.9× faster with a single-threaded processing time of 650 seconds compared to TensorFlow's 1905 seconds with eight threads at the 20.5 MB sample size. In other words, it pays off to use the less scalable but more efficient implementation in NumPy instead of the native implementation in TensorFlow.

(3) Random file access performance can affect the speedup. We have already discussed the impact of concatenation in Sec. 4.1 **(1)** and how random file access can hinder achieving high throughput and bandwidth utilization. By running the multi-threading experiments with system-level caching enabled, we can isolate the effect of random file access on speedup. For example, the `unprocessed` strategy of the MP3 pipeline (Fig. 12k) has a speedup of 2× when reading from storage with eight threads, versus a speedup of 6× when reading from memory (Fig. 12l). This shows that decoding does in fact scale well. The same effect changes the speedup of the FLAC pipeline from 4× (Fig. 12m) to 6× with eight threads (Fig. 12n).

4.5 Shuffling

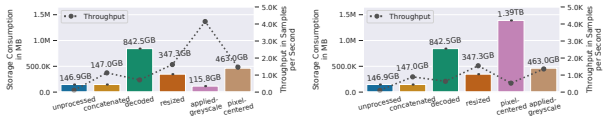
A common technique in DL is to change the order of the dataset in every epoch, so the optimizers do not see the same gradients in mini-batches. There are a few different approaches to shuffling the dataset, which include sampling from the dataset *with-* or *without replacement* [15, 32, 94]. Irrespective of which algorithm is used to modify the order of the dataset, it is a very memory-intensive problem, as the entire dataset has to be loaded into RAM. One solution is to create a buffer that fits into the memory and use a *with-replacement* sampling strategy to iterate over the entire dataset in a pseudo-random fashion [28], similar to reservoir sampling [88].

We implemented and profiled this approach with multiple sample counts, which confirms the naive assumption that the *per-sample* processing time for shuffling is constant. That means that shuffling has a linear relation to storage consumption and is not specific to a pipeline or dataset. The difference in per-sample processing time between shuffling and not shuffling for each sample size is (±0.5) 9.6ms on average. An additional characteristic is that the initial call to allocate a buffer is amortized with a bigger sample size, which manifests itself in the increasingly faster per-sample time with incremented sample sizes.

PRESTO's profiling can help to find the optimal place in the pipeline where shuffling should be applied. As the storage consumption of a strategy does not affect the runtime of shuffling, we do not recommend making shuffling part of the strategy selection. However, once a strategy is determined, we suggest to shuffle after the online pipeline step that yields the smallest data size. If we consider a fixed-size buffer for shuffling, the highest number of samples can be fit into the buffer when the size of the data sample is smallest. The higher the number of samples in the buffer, the higher the entropy; this, in turn, leads to a better approximation of the "true" gradient [44, 60].

4.6 Modifying the Pipeline

We added an additional preprocessing step to the CV pipeline to showcase how the trade-offs can shift in an already profiled pipeline. We decided on adding a step that converts images from RBG to greyscale because this is a common preprocessing step that affects the storage consumption and is not obviously compute intensive. To evaluate how an additional step will affect the pipeline performance, we profiled two setups: before and after the `pixel-centered` strategy.



(a) Before Pixel Centering (b) After Pixel Centering

Figure 14: Storage consumption (left y-axis) and throughput (right y-axis, dotted line) comparison of adding a greyscale transformation before and after the pixel centering.

Before discussing the results, we explain the characteristics of the new `applied-greyscale` strategy and compare it to `resized`. Converting a 3-channel image to greyscale should decrease the storage consumption by 3×, because we only need a single channel with the same datatype. The `resized` strategy reduces the size by 2.4× for the CV dataset, which is dependent on the average image resolution. Adding greyscaling will also affect the throughput of `pixel-centered` as the final storage consumption will be reduced as well.

The results in Fig. 14 show the effect of the additional greyscale step on the storage consumption and throughput. First of all, applying the greyscale step before the `pixel-centered` strategy increases the maximum throughput of the pipeline by 2.8×, from 1513 SPS with

`resized` (Fig. 14b) to 4284 SPS with `applied-greyscale` (Fig. 14a). While the last strategy of both setups performs similary, applying preprocessing steps that reduce the storage consumption consecutively increases the throughput for intermediate strategies. We additionally evaluated a setup where the resize and greyscale steps are interchanged before the pixel centering, but there was no significant difference in performance to Fig. 14a. The second setup with the last strategy `applied-greyscale` increases the throughput from 534 SPS at `pixel-centered` to 1384 SPS by reducing the data size from 1.4 TB to 463 GB. This supports our observation **(2)** from Sec. 4.1 that steps which reduce storage consumption should be investigated with priority when searching for the best performing.

5 LESSONS LEARNED

We find it important to summarize our findings more generically for both DevOps and ML practitioners so that they can use the PRESTO library and the generated insights for future analysis. These lessons are based on the analysis in Section 4.

(1) Storage consumption is an important characteristic when estimating throughput. We gravely underestimated the effect of storage consumption before conducting this study. The impact of storage consumption is a multi-faceted one, as it affects the storage hardware, its interconnects, the deserialization process and multi-threading capabilities. A small total storage consumption performs best if not throttled by a CPU bottleneck, and steps that reduce data size should be prioritized when searching for the best performing strategy. However, small sample sizes (≤ 0.08 MB) increase the online processing time dramatically irregardless of reading from storage or from memory and can be a reason for an underutilized I/O bandwidth. These two observations combined are the reason why fully preprocessed datasets did not yield the best throughput in 4 (CV, CV2-PNG, CV2-JPG, NLP) out of 7 pipelines.

(2) Multi-threading usually improves throughput, but the speedup can be limited for various reasons. Parallel execution of a pipeline is not a silver bullet when trying to speed up preprocessing. First of all, various issues can impede parallel speedup, such as calling external Python libraries or dealing with extremely short-running preprocessing tasks at small sample sizes. But even when parallel speedup of a strategy is reasonably good, a different strategy with a lower data volume to be read from storage may perform much better.

(3) It is recommended to use application-level caching whenever possible. Whenever the dataset fits into memory, application-level caching increased the throughput in our experiments by up to 15× with a high sample size. Application-level caching improved the throughput compared to system-level caching by a factor of 1.3-4.6×, and should be preferred as the deserialization of cached files can slow down the pipeline.

(4) Compression can be useful when not facing a CPU bottleneck. Compression can increase the throughput by a factor of 1.6-2.4× under few conditions: a high enough space saving of 73-93% and the absence of computationally expensive processing steps. However, estimating the space saving, as well as the decompression time is hard. Additionally, applying compression can increase the offline processing time between 1.1× and 13.5× compared to no compression. The overheads of compression should be taken into account and carefully weighted against I/O savings.

6 RELATED WORK

I/O profiling was already done in a micro-benchmark for TensorFlow by Chien et al. [13] which focused on different file systems and the check-marking functionality. In their experimental setup with AlexNet [49], the training data prefetching eliminated the effective preprocessing time, similar to our CV pipeline with the `unprocessed` strategy.

OneAccess, a unified data loading layer, functions as middleware for preprocessing and helps to run ML jobs on multiple nodes more efficiently by removing duplicate processing for hyperparameter tuning [42]. We have observed similar results where packing the dataset helps by allowing sequential data access, but their preprocessing seems to be done fully offline. Their sample lifecycle concept, which plans to store the data for a certain amount of time in anticipation of re-use, is a perfect fit for PRESTO's strategy optimization to select the lowest storage consuming dataset representation.

An example of improved I/O efficiency in DL for high-performance computing (HPC) is the framework DeepIO by Zhu et al. [97] which optimizes data loading to improve the training throughput. This framework could be used to complement our methodology to mitigate I/O bottlenecks.

Model throughput, which is coupled with GPU utilization, has been identified as an essential topic in the MLOps community [37, 59, 65]. Microsoft pointed out in a study [37] that underutilization of GPUs in multi-tenant settings is a problem for cloud providers. They evaluated how job locality and human errors can lead to unnecessarily idling resources. IBM implemented their FUSE-based [82] file system for object storage to improve the I/O loads in their IBM Fabric for Deep Learning services [65]. Additionally, they deployed caching mechanisms to improve long-term read throughputs if the dataset fits into memory. These studies touch on different pain points of the cloud providers as they start to recognize the potential of improving the deployment and resource usage of end-to-end DL pipelines, which integrate the previously overlooked preprocessing phase.

Another work in that direction is a recent NVIDIA study [63] which profiled the end-to-end training and inference time for multiple frameworks and models on their GPU and TPU servers. They highlight the benefits of different hardware solutions but do not take the preprocessing pipeline into account as they preprocess the dataset only once completely.

Relocating the preprocessing to more specialized hardware can increase performance and adds additional resources to the preprocessing profiling. NVIDIA's Data Loading Library (DALI), a Python library [61], provides common preprocessing steps for images, video, and audio formats, which can be used as a drop-in replacement for native pipelines that can be executed on the GPU. DALI has shown to improve the performance of multiple end-to-end DL pipelines [58, 62]. PRESTO can be applied on a DALI-enhanced pipeline, and while this may shift the trade-offs, it is essential to note that additional resources also introduce complexities like bandwidth restrictions, limited memory sizes, and in this case, double-use for preprocessing as well as training. DL models are stored in GPU memory for forward and backward passes, interfering with the improved preprocessing execution due to restricted memory size and computational capabilities when

executed in a pipelined fashion. The cost and ubiquity of CPU processing should be weighed carefully against GPUs and when in doubt, be optimized in an end-to-end manner with tools like CoorDL [58].

Our work focused on the trade-offs between storage consumption, throughput, and preprocessing time, while Mohan et al. [58] analyzed different types of stalls and focused on dividing the entire end-to-end DL pipeline into data fetches, preprocessing rate, and GPU processing rate. They have shown how to efficiently use the OS-level cache to improve fetch stalls while increasing the total time-to-accuracy on two 24 core machines with 8 GPUs and 500 GB RAM with an HDD and an SSD local storage. While our focus was more on consumer-level hardware, which does not allow this level of caching, we had a similar observation that the decoding step in CV is very inefficient and that slow preprocessing can bottleneck the training performance on the GPU. We provide a solution to one of their discussion points on mitigating the increased storage consumption due to decoding with a suitable strategy, which can cache the entire dataset even more efficiently combined with CoorDL.

Relocating the preprocessing onto an accelerator is also done in SMOL, a system that prepares the most efficient strategy on how to preprocess visual data *and* train a model in an end-to-end fashion while keeping the accuracy fixed [43]. We reproduced similar preprocessing bottlenecks regarding our image pipeline. However, while SMOL uses data compression steps and other techniques to speed up the data processing while providing the same model accuracy, we focused entirely on the preprocessing pipeline. Some of our insights can be incorporated into SMOL, such as partially preprocessing a pipeline for a specific set of hardware, allowing better throughput based on the presence of hardware en-/decoders or having additional compression in the preprocessing pipeline to increase the final throughput. Supplementing SMOL with our analysis of common preprocessing steps could enable it to work on non-image data.

7 DISCUSSION

While our analysis provides some key insights about how to profile and configure a preprocessing pipeline, we want to highlight some settings which could benefit from further research.

Datasets can grow over time. The results from PRESTO when profiling a pipeline and a static dataset should provide valuable insights if the dataset grows in the future. One exception is when the data representation of the newly added data is not compatible with the previous dataset, e.g., adding 4k images to a VGA-resolution dataset, which may slow down parts of the pipeline in unpredicted ways and result in different trade-offs. TensorFlow Extended (TFX) [9] is an ML platform to train and deploy models, which can be used to keep track of this shift in the dataset.

Storage bandwidth has shown to be a bottleneck for other similar MLOps studies [43, 58, 59]. We have shown that compression is a promising tool to mitigate storage-related bottlenecks but its efficacy is limited. Compression that is optimized to store *tensor-like* data could potentially provide even better throughput and space saving. Our recommended strategies from CV and NLP are integer tensors, but NILM, MP3, and FLAC use floating-point tensors with 32 and 64 bit, which suggests that different compression algorithms have to be considered depending on the data representation [51, 54]. When applied carelessly, compression can have severe effects on the entire online processing. PRESTO can be used to study the effect of compression in more depth.

Distributed computing for preprocessing. A common solution to speed up the execution jobs is using multiple worker nodes with frameworks like Apache BEAM [5] or Spark [95]. Preprocessing a dataset is a trivially parallelizable task by splitting the dataset into equal chunks for every worker to process simultaneously, except for shuffling or similar global dataset operators. While it is easy to follow PRESTO's recommendation and apply the offline transformation steps until the desired data representation is met, there are more complexities involved, like the data locality to workers, the locality of the workers to the training process, the amount of workers available, the interconnects, and the scheduling algorithm that supervises the job execution. This distributed setting will benefit from PRESTO's analysis, as storage consumption is correlated with the network bandwidth usage, and finding a strategy that has a good speedup will be even more effective with multiple workers. These insights may help to improve data management and scheduling. Nevertheless, additional profiling should be done to further optimize the pipeline execution for the specific cluster-computing framework.

Applicability for concurrent training. When considering a setup with a shared preprocessing pipeline between multiple distributed training jobs, such as in hyperparameter tuning, all of our insights are applicable, as the throughput T_4 can be fanned out to all training jobs. However, this setup adds load onto the network between the preprocessing node and the training nodes, which would not happen when running the preprocessing locally on the same machine that performs training. If the network can not handle the duplicated load of fanning out the preprocessed data per training job, it will become a new bottleneck.

8 CONCLUSIONS

This paper presents an analysis of seven concrete DL pipelines based on their typical preprocessing steps from CV, NLP, NILM, and the Audio domain. We provide a profiling library, PRESTO, that helps with detecting bottlenecks and automatically decide which preprocessing strategy is the most efficient based on an objective function. We show that not preprocessing the dataset before training is never the best solution for all pipelines, and fully preprocessing can affect the final preprocessing throughput negatively due to problems relating to I/O and storage consumption. Alternatively, we propose different strategies that increase the CV pipeline throughput by 3× and NLP by 13× while reducing their storage consumption compared to the fully preprocessed dataset. We provide insights into how storage consumption, different caching level and compression affect the preprocessing pipeline and how they can pinpoint where bottlenecks are formed. While multi-threading could be an effective way to speed up preprocessing, we show that using an intermediate preprocessing strategy is significantly more impactful to reduce processing time. Finally, we provide an intuition about profiling the preprocessing pipelines effectively by summarizing the generated insights to mitigate future bottlenecks in deep learning pipelines.

Acknowledgements. This work is funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 392214008.

REFERENCES

[1] Ellie Pavlick Stefanie Tellex Aaron Gokaslan, Vanya Cohen. 2019. OpenWebText Corpus. http://Skylion007.github.io/OpenWebTextCorpus. (2019).

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conf. on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, USA, 265–283.

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep Speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*. PMLR, USA, 173–182.

[4] Kyle Anderson, Adrian Ocneanu, Diego Benitez, Derrick Carlson, Anthony Rowe, and Mario Berges. 2012. BLUED: A fully labeled public dataset for event-based non-intrusive load monitoring research. In *Proceedings of the 2nd KDD Workshop on Data Mining Applications in Sustainability (SustKDD)*, Vol. 7. ACM, ACM, USA, 1–5.

[5] Apache 2021. *Apache Beam*. Apache. https://beam.apache.org/.

[6] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M Tyers, and Gregor Weber. 2019. Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670* (2019).

[7] Michele Banko and Eric Brill. 2001. Scaling to very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 26–33.

[8] Karim Said Barsim, Roman Streubel, and Bin Yang. 2014. Unsupervised adaptive event detection for building-level energy disaggregation. *Proceedings of Power and Energy Student Summit (PESS), Stuttgart, Germany* (2014).

[9] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 1387âĂŞ1395. https://doi.org/10.1145/3097983.3098021

[10] Enrico Blanzieri and Anton Bryl. 2008. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review* 29, 1 (2008), 63–92.

[11] Jamie Callan, Mark Hoy, Changkuk Yoo, and Le Zhao. 2009. *Clueweb09 Dataset*. http://lemurproject.org/clueweb09/.

[12] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005* (2013).

[13] Steven Wei Der Chien Chien, Stefano Markidis, Chaitanya Prasad Sishtla, Luis Santos, Pawel Herman, Sai Narasimhamurthy, and Erwin Laure. 2018. Characterizing Deep-Learning I/O Workloads in TensorFlow. *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (PDSW-DISCS)* (Nov 2018). https://doi.org/10.1109/pdsw-discs.2018.00011

[14] Common Crawl Foundation 2012. *Common Crawl*. Common Crawl Foundation. https://commoncrawl.org/.

[15] Christopher M De Sa. 2020. Random Reshuffling is Not Always Better. *Advances in Neural Information Processing Systems* 33 (2020).

[16] 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*. Ieee, 248–255.

[17] P. Deutsch. 1996. RFC1952: GZIP File Format Specification Version 4.3. (1996).

[18] P. Deutsch and J.-L. Gailly. 1996. RFC1950: ZLIB Compressed Data Format Specification Version 3.3. (1996).

[19] Egor Ershov, Alexey Savchik, Illya Semenkov, Nikola Banić, Alexander Belokopytov, Daria Senshina, Karlo Koščević, Marko Subašić, and Sven Lončarić. 2020. The Cube++ Illumination Estimation Dataset. *IEEE Access* 8 (2020), 227511–227527.

[20] Alexey Kopytov et al. 2021. *sysbench*. https://github.com/akopytov/sysbench.

[21] Jens Axboe et al. 2021. *FIO - Flexible I/O Tester*. https://github.com/axboe/fio.

[22] Mark Everingham, Luc Van Gool, Chris K. I. Williams, John Winn, and Andrew Zisserman. 2009. *The PASCAL Visual Object Classes Challenge 2009 (VOC2009) Results*. http://www.pascal-network.org/challenges/VOC/voc2009/workshop/index.html.

[23] Mark Everingham and John Winn. 2011. The pascal visual object classes challenge 2012 development kit. *Statistical Modelling and Computational Learning, Tech. Rep* 8 (2011).

[24] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2004. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 Conference on Computer Vision and Pattern Recognition Workshop*. IEEE, 178–178.

[25] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. 2004. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*. Springer, 97–104.

[26] Google 2008. *Protocol Buffers*. Google. http://code.google.com/p/protobuf/.

[27] Google 2020. *Better Performance with the tf.data API*. Google. https://www.tensorflow.org/guide/data_performance.

[28] Google 2021. *TF Dataset API - Shuffle*. Google. https://www.tensorflow.org/api_docs/python/tf/data/Dataset.

[29] Griffin Gregory, Holub Alex, and Perona Pietro. 2007. Caltech-256 object category dataset. *California Institute of Technology* 6 (2007).

[30] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[31] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep Speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).

[32] Jeff Haochen and Suvrit Sra. 2019. Random shuffling beats sgd after finite epochs. In *Int'l Conf. on Machine Learning*. PMLR, 2624–2633.

[33] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE International Conference on computer vision*. 2961–2969.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[35] Muhammad Huzaifah. 2017. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *arXiv preprint arXiv:1706.07156* (2017).

[36] Sebastian Jäger, Hans-Peter Zorn, Stefan Igel, and Christian Zirpins. 2018. Parallelized training of Deep NN: comparison of current concepts and frameworks. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*. 15–20.

[37] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2018. Multi-tenant GPU Clusters for Deep Learning Workloads : Analysis and Implications. *Tech. Rep.* (2018).

[38] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks.. In *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 1–13.

[39] Daniel Jorde, Matthias Kahl, and Hans-Arno Jacobsen. 2019. MEED: An Unsupervised Multi-Environment Event Detector for Non-Intrusive Load Monitoring. In *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 1–6.

[40] Daniel Jorde, Thomas Kriechbaumer, Tim Berger, Stefan Zitzlsperger, and Hans-Arno Jacobsen. 2020. CREAM, a Component Level Coffeemaker Electrical Activity Measurement Dataset. *Scientific Data* 7, 1 (2020), 1–13.

[41] Matthias Kahl, Anwar Ul Haq, Thomas Kriechbaumer, and Hans-Arno Jacobsen. 2017. A comprehensive feature study for appliance recognition on high frequency energy data. In *Proceedings of the Eighth International Conference on Future Energy Systems*. 121–131.

[42] Aarati Kakaraparthy, Abhay Venkatesh, Amar Phanishayee, and Shivaram Venkataraman. 2019. The case for unifying data loading in machine learning clusters. In *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*.

[43] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2020. Jointly Optimizing Preprocessing and Inference for DNN-Based Visual Analytics. (2020), 87–100. https://doi.org/10.14778/3425879.3425881

[44] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd Int'l Conf. on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conf. Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[45] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. 2017. OpenImages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages* (2017).

[46] Thomas Kriechbaumer and Hans-Arno Jacobsen. 2018. BLOND, a building-level office environment dataset of typical electrical appliances. *Scientific data* 5, 1 (2018), 1–14.

[47] Alex Krizhevsky, Geoffrey Hinton, et al. 2012. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (May 2012).

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*. Curran Associates Inc., USA, 1097–1105. http://dl.acm.org/citation.cfm?id=2999134.2999257

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural*

Information Processing Systems. Curran Associates, 1097–1105.
[50] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. 2015. Simultaneous feature learning and hash coding with deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3270–3278.
[51] Daniel Lemire and Leonid Boytsov. 2015. Decoding billions of integers per second through vectorization. *Software: Practice and Experience* 45, 1 (2015), 1–29.
[52] Jiajia Li, Mahesh Lakshminarasimhan, Xiaolong Wu, Ang Li, Catherine Olschanowsky, and Kevin Barker. 2020. A parallel sparse tensor benchmark suite on CPUs and GPUs. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 403–404.
[53] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*. Springer, 740–755.
[54] Peter Lindstrom and Martin Isenburg. 2006. Fast and Efficient Compression of Floating-Point Data. *IEEE transactions on visualization and computer graphics* 12 (09 2006), 1245–50. https://doi.org/10.1109/TVCG.2006.143
[55] Ruben Mayer and Hans-Arno Jacobsen. 2020. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. *ACM Comput. Surv.* 53, 1, Article 3 (feb 2020), 37 pages. https://doi.org/10.1145/3363554
[56] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *Int'l Conf. on Learning Representations*. https://openreview.net/forum?id=r1gs9JgRZ
[57] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[58] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. 2021. Analyzing and Mitigating Data Stalls in DNN Training. *VLDB Endowment* 14, 5 (2021), 771–784. https://doi.org/10.14778/3446095.3446100
[59] Derek G Murray, Jiri Simsa, Ana Klimovic, and Ihor Indyk. 2021. tf.data: A Machine Learning Data Processing Framework. *arXiv preprint arXiv:2101.12127* (2021).
[60] Aatila Mustapha, Lachgar Mohamed, and Kartit Ali. 2020. An Overview of Gradient Descent Algorithm Optimization in Machine Learning: Application in the Ophthalmology Field. In *Smart Applications and Data Analysis*, Mohamed Hamlich, Ladjel Bellatreche, Anirban Mondal, and Carlos Ordonez (Eds.). Springer Int'l Publishing, Cham, 349–359.
[61] NVIDIA 2018. *NVIDIA DALI*. NVIDIA. https://github.com/NVIDIA/DALI.
[62] NVIDIA 2019. *NVIDIA DALI Benchmarks*. NVIDIA. https://developer.nvidia.com/dali.
[63] NVIDIA 2020. *NVIDIA Data Center Deep Learning Product Performance*. NVIDIA. https://developer.nvidia.com/deep-learning-performance-training-inference.
[64] Lucas Ou-Yang. 2013. *newspaper library*. https://newspaper.readthedocs.io.
[65] Or Ozeri, Effi Ofer, and Ronen Kat. 2018. Object storage for deep learning frameworks. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*. 21–24.
[66] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: an ASR corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 5206–5210.
[67] Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. *English Gigaword Fifth Edition"*. https://catalog.ldc.upenn.edu/LDC2011T07.
[68] 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. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.
[69] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol. 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 2227–2237. https://doi.org/10.18653/v1/N18-1202
[70] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. 2019. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing* 13, 2 (2019), 206–219.
[71] Moo-Ryong Ra. 2018. Understanding the performance of ceph block storage for hyper-converged cloud with all flash storage. *arXiv preprint arXiv:1802.08102* (2018).
[72] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *OpenAI* (June 2018). https://s3-us-west-2.amazonaws.com/openai-assets/researchcovers/languageunsupervised/languageunderstandingpaper.pdf.
[73] Alec Radford, Jeffrey Wu, Dario Amodei, Daniela Amodei, Jack Clark, Miles Brundage, and Ilya Sutskever. 2019. *Better language models and their implications*. Technical Report. OpenAI. https://openai.com/blog/better-language-models.
[74] Yihui Ren. [n. d.]. Performance Analysis of Deep Learning Workloads on Leading-edge Systems. ([n. d.]). https://doi.org/10.1109/PMBS49563.2019.00017
[75] 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. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y
[76] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, 1715–1725. https://doi.org/10.18653/v1/P16-1162
[77] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).
[78] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.
[79] Roman A Solovyev, Maxim Vakhrushev, Alexander Radionov, Irina I Romanova, Aleksandr A Amerikanov, Vladimir Aliev, and Alexey A Shvets. 2020. Deep learning approaches for understanding simple speech commands. In *2020 IEEE 40th International Conference on Electronics and Nanotechnology (ELNANO)*. IEEE, 688–693.
[80] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.
[81] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2818–2826.
[82] Miklos Szeredi and et. al. 2018. *FUSE*. https://github.com/libfuse/libfuse.
[83] Antonio Torralba, Rob Fergus, and William T Freeman. 2008. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 11 (2008), 1958–1970.
[84] Kien Nguyen Trung, Eric Dekneuvel, Benjamin Nicolle, Olivier Zammit, Cuong Nguyen Van, and Gilles Jacquemod. 2014. Event detection and disaggregation algorithms for nialm system. In *Proceedings of 2nd International Non-Intrusive Load Monitoring (NILM) Workshop*.
[85] Han Vanholder. 2016. *Efficient Inference with TensorRT*. NVIDIA. http://on-demand.gputechconf.com/gtc-eu/2017/presentation/23425-han-vanholder-efficient-inference-with-tensorrt.pdf.
[86] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3156–3164.
[87] Abhinav Vishnu, Charles Siegel, and Jeffrey Daily. 2016. Distributed Tensorflow with MPI. *arXiv preprint arXiv:1603.02339* (2016).
[88] Jeffrey S. Vitter. 1985. Random Sampling with a Reservoir. 11, 1 (1985), 37âĂŞ57. https://doi.org/10.1145/3147.3165
[89] Gregory K Wallace. 1992. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics* 38, 1 (1992), xviii–xxxiv.
[90] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. 307–320.
[91] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 56 – 61. https://doi.org/10.25080/Majora-92bf1922-00a
[92] Wikipedia 2014. *English Wikipedia Corpus*. Wikipedia. https://www.english-corpora.org/wiki/.
[93] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. 2018. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992* (2018).
[94] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. 2021. Can Single-Shuffle SGD be Better than Reshuffling SGD and GD? (2021). arXiv:cs.LG/2103.07079
[95] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.
[96] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*. Springer, 818–833.
[97] Yue Zhu, Fahim Chowdhury, Huansong Fu, Adam Moody, Kathryn Mohror, Kento Sato, and Weikuan Yu. 2018. Entropy-aware I/O pipelining for large-scale deep learning on HPC systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 145–156.
[98] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on computer vision*. 19–27.
[99] Zhicheng Zhu, Shuai Zhang, Zhiqiang Wei, Bo Yin, and Xianqing Huang. 2018. A novel CUSUM-based approach for event detection in smart metering. In *IOP Conference Series: Materials Science and Engineering*, Vol. 322. IOP Publishing, 072014.

ACM Publishing License and Audio/Video Release

Title of the Work: Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines
Submission ID:moddm088

Author/Presenter(s): Alexander Isenko:Technical University of Munich;Ruben Mayer:Technical University of Munich;Jeffrey Jedele:Technical University of Munich;Hans-Arno Jacobsen:University of Toronto

Type of material:full paper

Publication and/or Conference Name: SIGMOD '22: International Conference on Management of Data Proceedings

1. Glossary

2. Grant of Rights

(a) Owner hereby grants to ACM an exclusive, worldwide, royalty-free, perpetual, irrevocable, transferable and sublicenseable license to publish, reproduce and distribute all or any part of the Work in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library, and to authorize third parties to do the same.

(b) In connection with software and "Artistic Images and "Auxiliary Materials, Owner grants ACM non-exclusive permission to publish, reproduce and distribute in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library.

(c) In connection with any "Minor Revision", that is, a derivative work containing less than twenty-five percent (25%) of new substantive material, Owner hereby grants to ACM all rights in the Minor Revision that Owner grants to ACM with respect to the Work, and all terms of this Agreement shall apply to the Minor Revision.
(d) If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

☑ A. Grant of Rights. I grant the rights and agree to the terms described above.

☐ B. Declaration for Government Work. I am an employee of the national government of my country/region and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are you a contractor of your National Government? ○ Yes ◉ No

Are any of the co-authors, employees or contractors of a National Government? ○ Yes ◉ No

---

3. Reserved Rights and Permitted Uses.

(a) All rights and permissions the author has not granted to ACM in Paragraph 2 are reserved to the Owner, including without limitation the ownership of the copyright of the Work and all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM in Paragraph 2(a), Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new ACM Consolidated TeX template Version 1.3 and above automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

NOTE: For authors using the ACM Microsoft Word Master Article Template and

Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

Please put the following LaTeX commands in the preamble of your document - i.e., before \begin{document}:

```
\copyrightyear{2022}
\acmYear{2022}
\setcopyright{acmlicensed}\acmConference[SIGMOD '22]{Proceedings of the 2022 International Conference on Management of Data}{June 12--17, 2022}{Philadelphia, PA, USA}
\acmBooktitle{Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12--17, 2022, Philadelphia, PA, USA}
\acmPrice{15.00}
\acmDOI{10.1145/3514221.3517848}
\acmISBN{978-1-4503-9249-5/22/06}
```

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

If you are using the ACM Interim Microsoft Word template, or still using or older versions of the ACM SIGCHI template, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA
© 2022 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-9249-5/22/06...$15.00
https://doi.org/10.1145/3514221.3517848

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library. Once you have your camera ready copy ready, please send your source files and PDF

to your event contact for processing.

---

4. ACM Citation and Digital Object Identifier.

(a) In connection with any use by the Owner of the Definitive Version, Owner shall include the ACM citation and ACM Digital Object Identifier (DOI).
(b) In connection with any use by the Owner of the Submitted Version (if accepted) or the Accepted Version or a Minor Revision, Owner shall use best efforts to display the ACM citation, along with a statement substantially similar to the following:

> "© [Owner] [Year]. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in {Source Publication}, https://doi.org/10.1145/{number}."

5. Audio/Video Recording

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ◉ Yes ○ No

6. Auxiliary Material

Do you have any Auxiliary Materials? ◉ Yes ○ No

* Your Auxiliary Materials Release is conditional upon you agreeing to the terms set out below.

[Defined as additional files, video or software and executables that are not submitted for review and publication as an integral part of the Work but are supplied by the author as useful resources.] I hereby grant ACM permission to serve files containing my Auxiliary Material from the ACM Digital Library. I hereby represent and warrant that my Auxiliary (software) does not knowingly and surreptitiously incorporate malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software.

☑ I agree to the above Auxiliary Materials permission statement.

☐ This software is knowingly designed to illustrate technique(s) intended to defeat a system's security. The code has been explicitly documented to state this fact.

7. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

◉ We/I have not used third-party material.
○ We/I have used third-party materials and have necessary permissions.

8. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part IV and be sure to include a notice of copyright with each such image in the paper.

◉ We/I do not have any artistic images.
○ We/I have any artistic images.

---

9. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants.

---

10. Enforcement.

At ACM's expense, ACM shall have the right (but not the obligation) to defend and enforce the rights granted to ACM hereunder, including in connection with any instances of plagiarism brought to the attention of ACM. Owner shall notify ACM in writing as promptly as practicable upon becoming aware that any third party is infringing upon the rights granted to ACM, and shall reasonably cooperate with ACM in its defense or enforcement.

---

11. Governing Law

This Agreement shall be governed by, and construed in accordance with, the laws of the state of New York applicable to contracts entered into and to be fully performed therein.

Funding Agents

1. Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) award number(s):392214008

DATE: **02/15/2022** sent to alex.isenko@tum.de at **13:02:38**

B. How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study

This paper is accepted as peer-reviewed conference paper at the *Proceedings of the VLDB 2024, Volume 17* and currently in the process of being published.

Reprinted with permission from

A. Erben, R. Mayer, and H.-A. Jacobsen. *How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study*. 2023. arXiv: 2306.03163 [cs.LG]

How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study

Alexander Erben
Technical University of Munich
alex.isenko@tum.de

Ruben Mayer
University of Bayreuth
ruben.mayer@uni-bayreuth.de

Hans-Arno Jacobsen
University of Toronto
jacobsen@eecg.toronto.edu

ABSTRACT

This paper aims to answer the question: Can deep learning models be cost-efficiently trained on a global market of spot VMs spanning different data centers and cloud providers? To provide guidance, we extensively evaluate the cost and throughput implications of training in different zones, continents, and clouds for representative CV, NLP and ASR models. To expand the current training options further, we compare the scalability potential for hybrid-cloud scenarios by adding cloud resources to on-premise hardware to improve training throughput. Finally, we show how leveraging spot instance pricing enables a new cost-efficient way to train models with multiple cheap VMs, trumping both more centralized and powerful hardware and even on-demand cloud offerings at competitive prices.

PVLDB Reference Format:
Alexander Erben,
Ruben Mayer, and Hans-Arno Jacobsen. . PVLDB, 17(1): XXX-XXX, 2023.
doi:XX.XX/XXX.XX
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at github.com/cirquit/hivemind-multi-cloud.

1 INTRODUCTION

Deciding whether to invest in on-premise hardware or move to the cloud for deep learning (DL) is not easy. Wanting to scale existing infrastructure means paying upfront, as combining cloud and on-premise is not an option with popular DL frameworks due to needing a dedicated high-bandwidth interconnect. To enabled model- and data-parallelism, current state-of-the-art accelerators have bandwidths of 900 GB/s for intra-node [18] and 25 Gb/s for inter-node setups [25, 38]. Due to the initial investment of the cloud providers in the accelerators, they naturally want to reap profit by maximizing resource utilization. Therefore, it is common to have "spot" pricing, which offers the VMs at a strongly reduced rate, typically at a 40-90% discount (Table 1), but with the drawback that the VM can be terminated at any time if another customer is willing to pay the on-demand price [32]. Unfortunately, popular DL frameworks have not been developed with failure semantics in mind and cannot adequately deal with peers that fail [11]. While services like Amazon Sagemaker [13] and projects like Skypilot [43] offer automatic job migration in case of VM termination, they are limited to single-node training due to the bandwidth requirements between accelerators.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 17, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

Type \ Cloud	GC	AWS	Azure
T4 Spot	0.180 $/h	0.395 $/h	0.134 $/h
T4 On-Demand	0.572 $/h	0.802 $/h	0.489 $/h
Traffic (inter-zone)	0.01 $/GB	0.01 $/GB	0.00 $/GB
Traffic (inter-region) US	0.01 $/GB	0.01 $/GB	0.02 $/GB
Traffic (inter-region) EU	0.02 $/GB	0.01 $/GB	0.02 $/GB
Traffic (inter-region) ASIA	0.05 $/GB	0.01 $/GB	0.08 $/GB
Traffic (inter-region) OCE	0.08 $/GB	0.01 $/GB	0.08 $/GB
Traffic ANY-OCE	0.15 $/GB	0.02 $/GB	0.08 $/GB
Traffic (between continents)	0.08 $/GB	0.02 $/GB	0.02 $/GB

Table 1: Average us-west cloud pricing in April '23.

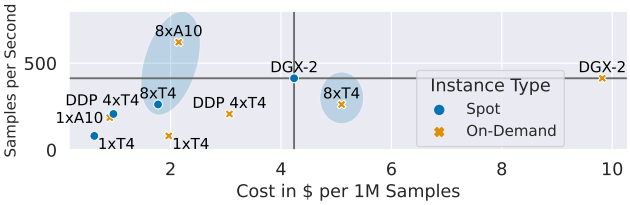


Figure 1: Cost to throughput tradeoff for ConvNextLarge at different instance types. Our training setups (circled) are cheaper (8xT4) and faster (8xA10) than the centralized offering (DGX-2).

But what if we could use spot pricing for long-running, distributed jobs and reduce bandwidth requirements to leverage multiple low-cost GPUs? This could be possible through a framework for collaborative DL training, Hivemind [39], which inherently deals with peers that can stop running at any time. While there is research on how Hivemind can be used for training on spot VMs [16, 36, 37], it does not compare the cost-throughput tradeoff for different cloud offerings or perform ablation studies on geographic distribution or model sizes.

To motivate this new possibility, we trained the ConvNextLarge model [28] on the Imagenet1K dataset [14] on different Google Cloud hardware (T4's and DGX-2), and on the very competitively priced A10 from LambdaLabs (see Section 6 for the full experimental description). Figure 1 shows the training throughput and the costs per 1 million processed samples for each setup. The single node (1xT4, 1xA10, DGX-2) experiments show the current state-of-the-art cost-throughput ratio for training on GC and LambdaLabs. The DGX-2 node is the fastest, with a throughput of 413 SPS, but it also costs $6.30/h ($4.24/1M samples), shown by the horizontal and vertical lines. The single-accelerator experiments (1xT4, 1xA10) have a better cost-throughput ratio ($0.62/1M samples and $0.9/1M samples), but have a much lower throughput of 80 and 185 SPS, respectively. However, when using our approach of distributing the training between multiple GPUs with Hivemind (circled), we make training possible that is both faster (8xA10, 621 SPS, $2.15/1M samples) and cheaper (8xT4, 262 SPS, $1.77/1M samples) than using the DGX-2. Every cloud

provider deals differently with how they price spot instances and network traffic (cf. Table 1) and has varying interruption rates for different accelerators [22]. Being able to choose the best option was not possible before, and having the option to combine older, more available GPUs is a net benefit for both consumers and cloud providers alike.

We aim to develop guidelines and help practitioners assess under which conditions they can cost-efficiently speed up their training tasks with spot instances. To be able to do this, they need a precise definition of the model size at which geo-distributed spot training becomes viable, what hardware can be used for it, and what the minimum bandwidth and latency are. We close this research gap by performing a comprehensive analysis of multiple DL tasks from CV and NLP, breaking down how time is spent in each epoch, and comparing them to non-distributed runs to quantify the advantages and disadvantages of distributed spot training. We determine which models scale with additional spot instances and which cannot be scaled without running into a communication bottleneck or resource inefficiencies. To quantify total training cost, we assess cost-effectiveness and evaluate a hybrid or multi-cloud approach with popular cloud providers through training on up to four continents. For comparison of the models' scalability and to show which of them can be trained in a distributed fashion, we introduce the *granularity metric*, the ratio of calculation to communication time, and show how it can be used for predicting performance with different hardware setups. Finally, we summarize our lessons on how to design geo-distributed spot training and what to watch out for when evaluating the feasibility of such a training regime. Our contributions are:

(1) **We analyze the impact of multi-cloud training with spot and on-demand instances from Google Cloud (GC), Microsoft Azure, Amazon Web Services (AWS), and LambdaLabs on cost-efficiency.** While we find performance penalties due to remote versus on-premise compute resources, the throughput still scales with increased computing power. By leveraging multiple spot instances with one T4 GPU each, we can be more cost-efficient than a DGX-2 node or the very competitively priced A10 offerings from LambdaLabs.
(2) **We investigate the suitability of geo-distributed training for various CV and NLP models and hardware configurations on up to four continents.** Not surprisingly, the more parallelizable and the larger the task, the better the performance. Moreover, we verify the scalability claims of the related work and define additional constraints, such as the minimum granularity for effective training. This enables, for the first time, distributed training of smaller million-parameter models (12M-560M) over <1 Gb/s bandwidth and >150ms latency networks.
(3) **We evaluate two different hybrid-cloud experimental setups with consumer- and server-grade on-premise hardware** and try to improve the throughput with a bandwidth of, at worst, 50 Mb/s to the cloud resources. While we show that it is possible to improve throughput even at these constraints, local cloud offerings are better suited for models that show limited suitability for distributed training.
(4) We summarize our findings of training in a geo-distributed, multi-cloud environment. **We propose the granularity metric to compare model suitability for distributed spot training** and estimate training performance with additional spot VMs. This provides guidance on the trade-off between performance and cost when using geo-distributed spot instances. To apply our findings, we perform a case-study on a state-of-the-art model from the ASR domain and achieve speedups on low-end hardware.

2 DEEP LEARNING ON SPOT INSTANCES

In this section, we describe how the Hivemind framework works and how it can enable distributed spot training.

2.1 Hivemind

Hivemind [39] is a PyTorch-based [31] framework developed initially to enable collaborative DL training where participants could donate their heterogeneous hardware to train a single model together in a data-parallel fashion. Its main difference to other state-of-the-art distributed training frameworks, such as PyTorch DDP [25] and DeepSpeed [34], is that it runs in a decentralized fashion and can handle peers that drop out at any stage of the training. It does so with two features: a distributed hash table [30] (DHT) which spans over all participating peers for metadata storage, such as training progress and peer health, and a gradient averaging algorithm that is designed to reduce the impact of lost gradients. A key difference to other distributed training frameworks is the definition of a *hivemind epoch*, which is the number of samples that must be aggregated before an averaging step is performed. This sample count is called the *target batch size* (TBS), which corresponds to the minibatch size in standard DL training. The DHT is used for coordination, and shortly before the TBS is predicted to be reached, the peers start to form the initial groups for averaging. The time allocated for group forming is called *matchmaking time* and typically runs asynchronously to the training (cf. Section 3). The individual peer gradients are accumulated locally and sent to the other peers via an adaptive all-reduce algorithm (MoshpitSGD [37]). The next hivemind epoch starts after each peer applies the accumulated gradients to the local model. The advantage of Hivemind for geo-distributed training comes from cumulating different techniques, such as Delayed Parameter Updates [35], big-batch training [44] and aggressive communication quantization [15]. All of these combined reduce time and frequency of the communication rounds, which in turn makes training on heterogeneous devices and low-bandwidth networks possible.

2.2 Distributed Spot Training

In this paper, we focus only on models that fit into the memory of a single GPU, as we are interested in utilizing data parallelism on cheaper and more readily available hardware. However, our insights are applicable to larger models with techniques such as ZeRO offloading [35], more aggressive quantization [41] and even model parallelism [36]. The current options for data parallelism are either using multiple GPUs on the same node (e.g., a DGX system with eight GPUs) or having multiple nodes with a GPU each in the same high-bandwidth network (>25 Gb/s) to minimize communication time. The latter does not work on cheap but interruptable instances, while the former has some use in the form of Amazon Sagemaker but is limited to a single node and is typically very pricey (spot pricing for DGX-2 is $6.30/h versus 8xT4 at $0.72/h on GC). However, using Hivemind, a new training scenario becomes feasible: Distributed training in a decentralized fashion on interruptable VMs with bandwidths of

<1 Gb/s. Since spot instance prices change hourly depending on the time of day and zone availability [22], and can vary widely between cloud providers (cf. Table 1), training between continents and in multiple clouds could potentially be more cost-effective than using a single, more computationally powerful node at spot prices.

With the newly added training setups from Figure 1 (circled), it was not previously possible to choose the best option, and having the option to combine older, more available GPUs is a net benefit for both consumers as well as cloud providers. Our paper shows that it is possible to train on multiple clouds across multiple continents and provides guidelines on how to accomplish this cost-efficiently.

3 MODEL SUITABILITY

Model	Params
ResNet18 [20]	11.7M
ResNet50 [20]	25.6M
ResNet152 [20]	60.2M
WideResNet101_2 [46]	126.9M
ConvNextLarge [28]	197.8M

(a) CV

Model	Params
RoBERTaBase [27]	124.7M
RoBERTaLarge [27]	355.4M
RoBERTaXLM [12]	560.1M

(b) NLP

Table 2: Model suitability parametrization.

Selecting suitable models with a big enough parallel workload is essential to ensure successful distributed spot training. To cover a wide range of established models, we drew from MLCommons' comprehensive DL training benchmark [29]. We used models from the CV and NLP domains and gradually increased their size and TBS to increase the parallel compute amount (Table 2). As discussed in Section 2, the TBS may be exclusively responsible for the success of distributed training and was chosen to cover both medium and large batches (8K, 16K and 32K). These minibatch sizes start to become more common due to the LAMB optimizer [44], which works well enough for both smaller (512) and huge batches (64K) and should be representative of state-of-the-art workloads. For a represenatative experimental study with a minibatch size of 256 on the automatic speech recognition model (Whisper [33]), please refer to Section 11. All experiments were run with FP16 precision, as the target T4 GPUs have a considerable improvement in FLOPs compared to FP32 (8:1).

For **CV**, we take five models from the extended ResNet family, starting with the smallest one, ResNet18 (RN18), ResNet50 (RN50), ResNet152 (RN152), WideResNet101_2 (WRN101) and ConvNext-Large (CONV), which is almost 20 times larger than RN18. These models were popularized due to their ability to help with the vanishing gradient problem by using residual connections between layers. Currently, they are not only used for classification, but can serve as an embedding of images by removing the classification head [17, 40]. For the dataset, we use Imagenet1K [14] and train the classification task, which tries to assign one of 1000 classes to each image.

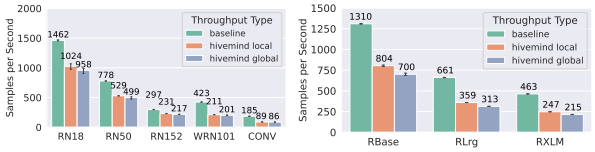
For **NLP**, we selected three models from the BERT family: RoBERTaBase (RBase), RoBERTaLarge (RLrg), and RoBERTaXLM (RXLM). We used the same configuration as the original models and trained them on masked language modeling, a common pre-training task. RoBERTa models were a replication study of BERT but with a focus on better hyperparameter tuning, leading to state-of-the-art results and proposed using much higher minibatch sizes than previously common. The text dataset is March '22 Wikipedia [19].

When we run our experiments in a multi-cloud environment on spot instances, we cannot plug in proprietary cloud storage or wait for the dataset to download, as the instances can be terminated anytime. To simulate a real-world deployment with a non-public dataset, we chose an independent S3 storage provider, Backblaze (B2) [3]. Backblaze has replicated data centers that can better serve requests from anywhere worldwide, guaranteeing a reasonable ingress rate from every continent. Additionally, the cost is very manageable at $0.01/GB rate for egress and $0.005/GB/month for storage. A detailed analysis of the costs incurred for the experiments can be found in Section 5. We access the datasets on-demand via shards in the `tar` format with the WebDataset library [9]. We chose WebDataset due to its features like automatic local caching, streaming decompression, streaming preprocessing, and having an easy to work with archive format that allows representing the data in its original format. Finally, for the Hivemind parameterization, we enabled delayed parameter averaging (DPU) [35] to enable simultaneous gradient communication and computation at the expense of a round of staleness. We selected FP16 compression for peer-to-peer communication.

Experimental design. First, we must verify that our models are suitable for cloud training. For this purpose, we evaluate them on the powerful Ampere GPUs first - if they scale there without facing a communication bottleneck, they should also scale on the slower T4, which is common at GC, AWS, and Azure. We use the LambdaLabs [7] for these experiments, which gives us on-demand A10 GPUs for just $0.60/hour, but currently offer their services only in the US West region. All experiments are performed on the 515.65.01 driver, CUDA 11.6, and PyTorch 1.13.1. We profiled a network bandwidth of 3.3 Gb/s and a latency of 0.3 ms between the Lambda VMs.

To establish a fair baseline, we train all models from Table 2 on a single GPU that achieves large minibatch sizes through gradient accumulation. Processes logs system metrics every second and evaluates the training performance whenever a batch is processed. Finally, all multi-GPU experiments are monitored with a training monitor that scrapes the DHT every second to log the peer state and training progress synchronously.

(1) Hivemind penalty. Using Hivemind as middleware to share gradients and keep a fully decentralized architecture running harms performance compared to single-node training. We can compare the effects of Hivemind training by looking at three metrics: *baseline*, the single GPU throughput, *hivemind local*, normalized GPU throughput without the averaging step, and *hivemind global*, the actual normalized GPU throughput. When comparing the baseline and local speed in Figure 2 for a setup with two GPUs, running Hivemind reaches at best 78% (RN152) and at worst 48% (CONV) of the baseline performance. Unsurprisingly, the larger the model size, the worse the penalty gets due to the increased size of the accumulated gradients (GAC) over each step. However, the baseline also applies gradient accumulation to reach the target minibatch size without the performance drop. After isolating the respective function calls, there seems to be a slight inefficiency in how GAC is implemented in Hivemind versus the native PyTorch call. We are working with the maintainers to fix this issue [6]. On the other hand, the disadvantage of synchronization is minimal under the perfect conditions of a good interconnect. The global speed in Figures 2a and 2b only degrades at best to 97% (CONV) to at worst to 87% (RBase) compared to the local throughput, meaning that the communication under these conditions only accounts for a fraction of the total training time. This degradation is inversely correlated to the model size due to larger



(a) CV (b) NLP

Figure 2: Hivemind penalty on normalized throughputs.

models training quadratically longer per parameter, but the communication only increases linearly [36]. While an implementation issue currently affects performance, and the worst total performance drop is at 47% (CONV baseline vs. global), scaling is still possible with a ratio of roughly 2:1 of GPUs to throughput. We further refine this ratio in the following section by comparing which models are most suitable to be trained in a distributed environment.

(2) Less suitable models for distributed spot training. While training billion-parameter NLP models scale well due to the "square-cube" law, the minimum model size is not yet fully defined [36]. The reason is that many factors play a role in whether a model is suited for geo-distributed training. On the one hand, a small model results in small gradients exchanged between peers, so the averaging step is fast. On the other hand, a small model will also reach the TBS faster than larger models, which may lead to a low speedup if the calculation time is disproportionally lower than the communication time. We found the granularity metric [21], typically used in high-performance computing, practical to attach a comparable value to each setup to quantify the ratio of the calculation and communication time (Equation (1)). The higher the granularity, the more parallelizable the task, as more calculation can be distributed between peers, ensuring a good speedup. It is important to note that this metric depends on the model and the hardware being used. The communication time is affected by the parameter count, and the calculation time is affected by the layer type of the parameters (including feedforward, convolution, and transformer). Therefore, the calculation time can decrease with improved hardware, which we evaluate in Section 6.

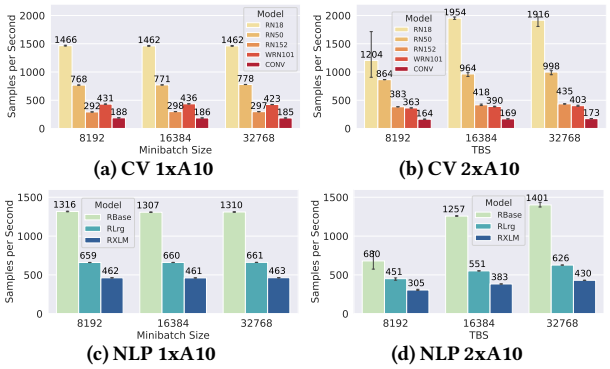
$$G = \frac{T_{\text{calc}}}{T_{\text{comm}}} \tag{1}$$

Another parameter that affects the calculation time is the TBS that all peers work to accumulate. There is a practical limit to the TBS where a model is still trainable, which is currently at 64K with the LAMB optimizer [44]. This limits the possibility of improving the speedup of small models by increasing the batch size, meaning that at some point, the speed will be limited by the communication time (Equation (2)).

$$\lim_{T_{\text{calc}} \to 0} \frac{\text{samples}}{T_{\text{comm}} + T_{\text{calc}}} = \frac{\text{samples}}{T_{\text{comm}}} \tag{2}$$

It is important to remember that just increasing the TBS to create more T_{calc} can have a grave effect on training performance if the optimizer is not adequately selected and configured.

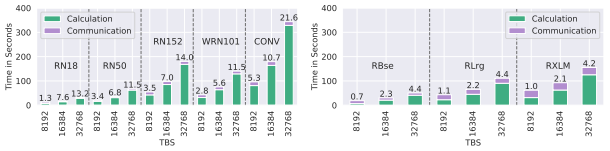
Our experimental results in Figure 3 show the practical implications of this observation. For the 2xGPU experiments in Figures 3b and 3d, we can see the effect of a TBS increase which improves the total throughput. Doubling the TBS equals cutting down the per-sample communication cost by two, which leads to the slight increase in performance visible in both CV and NLP experiments. However, the smallest models, RN18 and RBase, fluctuate significantly at a TBS of 8K due to a minimum matchmaking time of 5 seconds. Whenever all peers accumulate the TBS in less than 5 seconds, the asynchronous



(a) CV 1xA10 (b) CV 2xA10

(c) NLP 1xA10 (d) NLP 2xA10

Figure 3: Throughput comparison between single GPU baselines and the Hivemind runs with two GPUs.



(a) CV (b) NLP

Figure 4: TBS vs. total training time on 2xA10s. Granularity is shown above each bar. Dotted lines separate different models.

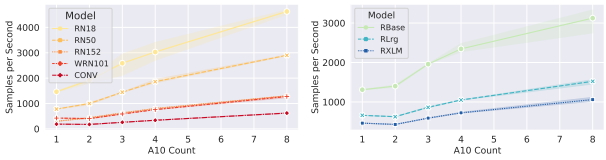
thread that matches the peers in groups to perform the all-reduce may still need to finish. This results in an unstable averaging time, which limits the scalability of small models with a small TBS.

To illustrate how the TBS and model size affect the individual timings, we visualize the total training time split up into the calculation and communication time in Figure 4. CV models are generally computationally more expensive and have a higher granularity than NLP models, which have slightly longer averaging rounds due to the much larger model sizes (cf. Table 2). When comparing the models at the same TBS (e.g., 32K), there is an inconclusive relation between runtime and parameter count. Some models increase their runtime with parameter count w.r.t. smaller models (RN50 to RN152, RBase to RLrg), while others decrease their runtime (RN152 to WRN101, RLrg to RXLM). This performance is due to not all layer parameters contributing similarly to computational complexity. Depending on the specific architecture, even models with more parameters can be faster to train due to a more efficient architecture, such as the WRN101 [46].

The communication time between different TBS sizes stays the same, barring the two matchmaking time exceptions (RN18, RBase), as the gradients are accumulated before being sent. For all other models, doubling the TBS leads to exactly double the amount of work and doubles the granularity. With a TBS of 32K, all models have a granularity of at least 4.2 (RXLM) and at most 21.6 (CONV), which show strong scaling potential. Therefore, we decided to use a TBS of 32K for all following experiments to ensure that the setup scales before introducing bandwidth and computational limitations.

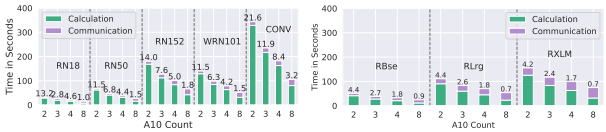
Summarizing, whether a model is scalable without network bandwidth limitations depends on the minimum time to reach the TBS and on the granularity. Tuning the TBS is possible to a certain extent but depends on the specific training task and optimizer configuration.

(3) Per-GPU speedup decreases with low granularity. To evaluate the scalability with additional hardware, we profile all models



(a) CV (b) NLP

Figure 5: Throughput comparison from 1 to 8 A10 GPUs.



(a) CV (b) NLP

Figure 6: Multi-GPU scalability at 32K TBS. Granularity is shown above each bar.Dotted lines separate different models.

on 2,3,4, and 8 GPUs with a TBS of 32K. Figure 5 shows the throughput for all models in the different hardware scenarios. Generally, all models scale well regardless of size, with the best speedup of 4.37x (RN152) and the lowest at 2.29x (RXLM) with 8 GPUs. There is a visible trend in the per-GPU contribution to the speedup ($\frac{\text{speedup}}{\#\text{GPUs}}$). The more GPUs we add, the lower the contribution, e.g., RN18 goes from 0.7 to 0.4 with two to eight GPUs, respectively. This decrease is likely to continue due to a granularity of 1.0 at 8 GPUs (Figure 6a), as doubling the GPUs would, at best, increase the throughput by 33% by halving the calculation time. However, the more computationally expensive the models are, the slower the per-GPU contribution falls off and the larger the granularity is (RN152, CONV). This does not hold true for our NLP models (Figure 6b); while they have increasingly more model parameters, the only difference between the two biggest models, RLrg and RXLM, is the vocabulary size increase of 50K to 250K. Due to how embedding layers are lookups, the forward pass is not affected by the increased embedding size, but the backward pass is. This results in a smaller increase of the calculation time while communication increases linearly with the number of parameters.

Additionally, we see the drop in throughput when comparing the single GPU and dual GPU experiments for most larger models (Figure 5), which stems from observation **(1)** of the Hivemind penalty.

We also observe that with each subsequent doubling of GPUs, the calculation time is halved, while the communication increases sub-linearly due to the more efficient group-based all-reduce of MoshpitSGD [37]. For example, the averaging step for the RXLM on 2xA10 takes 5 seconds per GPU (10s total), while the 8xA10 averaging step takes 1.8 seconds per GPU (14.4s total).

In summary, all models show a speedup but have a decreasing per-GPU contribution due to smaller granularity with more GPUs. Therefore, the larger the model and TBS, the greater the scaling potential. High granularity is a good indicator of scalability, and since the communication time only increases linearly with additional peers (cf. Section 2.1), knowing the initial calculation time is a good indicator of future throughput. Under the optimal conditions of good compute performance and an interconnect with relatively high bandwidth, scaling was not a problem. But what happens under less favorable conditions in geo-distributed settings?

Exp. Name	Resources	Total
A-{1,2,3,4,6,8}	{1, 2, 3, 4, 6, 8}xUS	1,2,3,4,6,8
B-{2,4,6,8}	{1, 2, 3, 4}xUS + {1, 2, 3, 4}xEU	2,4,6,8
C-{3,6}	{1, 2}xUS + {1, 2}xEU + {1, 2}xASIA	3,6
C-{4,8}	{1, 2}xUS + {1, 2}xEU + {1, 2}xASIA + {1, 2}xAUS	4,8

Table 3: Geo-distributed experiments on GC with T4 VMs.

4 GEO-DISTRIBUTED PERFORMANCE

As spot prices for the same hardware differ depending on the region, zone, and time of day [22], it might be a good idea to use VMs across different data centers. However, is the connectivity between regions and continents good enough to enable distributed deep learning? To explore this question, we decided to conduct three types of experiments (Table 3):

(A) Intra-zone Can we scale if the VMs are co-located in the same zone (`us-central-1`)?

(B) Transatlantic Can we scale when we combine VMs from two regions (US and EU), and what happens when the compute is unevenly distributed across regions?

(C) Intercontinental Can we scale if we combine VMs from four continents (US, EU, ASIA, AUS)?

Experimental design. Based on the insights from Section 3, we decided to use the largest models (CONV, RXLM) for all further cloud experiments in Sections 4 to 6, with the TBS of 32K as a baseline with good scaling properties. We abbreviate them with their respective domain names (CV, NLP). We used Google Cloud [4] for all experiments in this section, as they were the first to give us access to all necessary zones. The default networking solution in GC is the "Premium Tier", which tries to use a Google-owned network instead of the public internet. We measured the throughput and latency between all zones via `iperf` and `ping` and report the average of 5 consecutive runs in Table 4. Unsurprisingly, the diagonal shows that the local connectivity between zones runs at almost 7 Gb/s with a latency of 0.7ms, probably due to the hypervisors being in the same data center. While the up- and download were perfectly symmetrical in all setups, the throughput dropped to <210 Mb/s for all non-local connections. The US-based data center is located in Iowa and is best connected with at least 120 Mb/s to the remaining regions, namely Belgium in the EU (6,911km), Taiwan in ASIA (11,853km), and Sydney in Australia (AUS, 14,555km), presumably due to the physical distance. The lowest bandwidth and highest latency connections are between the EU region and ASIA and AUS, reaching around 80 Mb/s and 270ms. We decided to use the `n1-standard-8` template with eight cores, 30 GB memory, and a T4 GPU, as the smaller image with 15 GB was insufficient to meet the memory requirements for gradient application on the CPU with the biggest models. The experiment naming in this section is prefixed with the type of location **(A)**, **(B)** or **(C)** and the number of VMs, e.g., A-4 is the intra-zone experiment with 4 VMs. The full experimental description is specified in Table 3.

From \ To	US	EU	ASIA	AUS
US	6.90	0.21	0.13	0.12
EU	0.21	6.81	0.08	0.07
ASIA	0.13	0.08	6.79	0.16
AUS	0.12	0.07	0.16	6.84

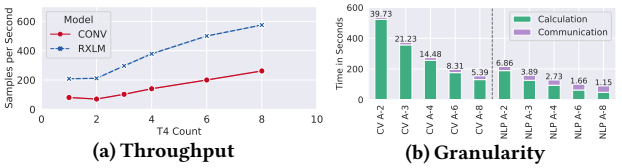
(a) Single stream TCP throughput in Gb/s.

From \ To	US	EU	ASIA	AUS
US	0.66	103.11	157.09	176.19
EU	103.14	0.65	253.10	271.98
ASIA	157.08	253.09	0.72	131.45
AUS	175.98	272.08	131.42	0.64

(b) ICMP latency in ms.

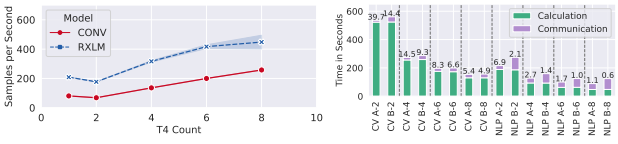
Table 4: Throughput and latency between GC zones.

(A) Intra-zone scalability. Figure 7 shows the result of the intra-zone experiments, which we used as a baseline to compare geo-distributed deployments to. As the scalability of the CV and NLP models was already shown with much better hardware and slightly worse



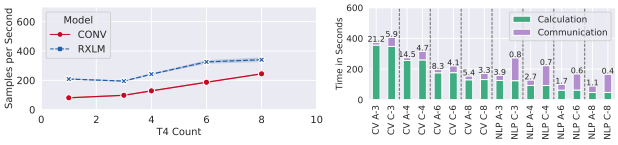
(a) Throughput (b) Granularity

Figure 7: (A) Intra-zone performance for CV and NLP.



(a) Throughput (b) Granularity

Figure 8: (B) Transatlantic performance for CV and NLP.



(a) Throughput (b) Granularity

Figure 9: (C) Intercontinental performance for CV and NLP.

network connectivity (cf. Section 3), the scalability with the T4 GPUs is not too surprising. We do not see an improvement in throughput for two GPUs for either model due to the Hivemind penalty discussed in Section 3. However, starting with three GPUs, we see an increase in throughput with a maximum speedup of up to 3.2x CV and 2.75x for NLP at eight GPUs. CV's per-GPU speedup ($\frac{\text{speedup}}{\#\text{GPUs}}$) is almost linear (0.43, 0.42, 0.43, 0.41, 0.41), while NLP starts dropping off faster (0.51, 0.47, 0.45, 0.40, 0.34) for 2, 3, 4, 6 and 8 GPUs, respectively. The reason for this is the NLP granularity of 1.15 with 8 GPUs indicating an almost equal part in communication and calculation (Figure 7b) due to the much longer averaging round related to the model size (198M vs. 560M parameters). The peak network bandwidth utilization between peers was at most a symmetric 1.1 Gb/s while averaging and 33 Mb/s ingress while training due to data loading. This means that the network bandwidth of 7 Gb/s was not a limiting factor.

(B) Transatlantic scalability. We scale when computing hardware is local. However, what happens when there is cheap capacity in another region? In this case, we study the throughput of experiments with resources in the `us-west` and `eu-central` regions (B-2,4,6,8).

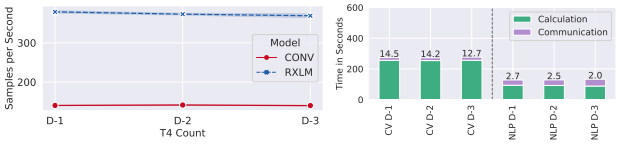
The B-2 experiment has one VM in the US and one in the EU, achieving a virtually identical throughput of 68.4 (US-EU) versus 70.1 (US) at CV (Figure 8a). Our maximum peak egress rate of 250 Mb/s does not affect the CV experiments, while the US experiments peaked at 1.1 Gb/s. The reduction in bandwidth penalizes NLP harder, where we are 16% slower with 177.3 SPS (US-EU) compared to the intra-zone experiment with 211.4 SPS (US). The resulting increased communication can be easily seen in the granularity analysis in Figure 8b (NLP A-2,4,6,8 vs. B-2,4,6,8). As only communication time increases in the NLP **(B)** experiments compared to **(A)**, a granularity of $\gg 1$ indicates good scalability: Adding two more GPUs to the B-6 experiment with a granularity of 1.03 results in a throughput increase of 15% (B-8) relative to the baseline. Meanwhile, adding two more GPUs to the B-2 experiment with a granularity of 2.21 results in a throughput increase of 77% (B-4) relative to the baseline.

In the B-4 experiment, we look at what happens when we increase the number of VMs to four, with two in the US and two in the EU. Nothing surprising happens with CV, as the workload continues to be mostly computation, with a throughput of 135.8 (B-4), only 3% slower than the intra-zone experiment with 140.4 SPS (A-4). However, at NLP, things get more interesting as we now have more overall communication with four peers, but they can average locally first and only later transmit across the Atlantic. However, compared to their A-counterparts, we do not see a difference in relative scalability with either B-4, B-6, or B-8. This means that training across regions **(B)** is slower, but the contribution per GPU decreases at the same rate as in training within a zone **(A)**. The per-GPU speedup with additional hardware reduces at the same rate for either setup (between 0.05 and 0.06). This results in two observations: First, communication overhead scales linearly with the number of peers. Second, we only have to pay the penalty for transatlantic training once. However, we cannot expect a significant improvement in communication efficiency when we increase the amount of available local resources.

Summarizing, with an transatlantic setup, CV achieves a virtually identical maximum speedup of 3.2x with 8 GPUs compared to A-1 (B-8 is 2% slower than A-8), while NLP is more affected by lower network bandwidth and only achieves a speedup of 2.15x (B-8 is 22% slower than A-8). The transatlantic training penalty is applied once; however, it does not affect the relative scaling with additional compute resources.

(C) Intercontinental scalability. To take geo-distribution to the extreme, we spawn VMs in up to 4 regions: USA, EU, ASIA, and AUS, to see how much worse bandwidth affects the training throughput (C-3,4,6,8 in Table 3).

How does the intercontinental penalty investigated in **(B)** affect deployments with a single GPU on each continent? Comparing the A-3 and C-3 experiments with three local versus three fully remote GPUs, CV is only 5% slower, while NLP suffers a 34% drop in throughput (Figure 9a) and does not even reach the baseline single GPU performance (A-1). The peak egress for each region was 318, 258, and 237 Mb/s for the US, EU, and ASIA, respectively. Since our bandwidth measurements were 210 and 130 Mb/s from the US to the EU and ASIA, respectively (Table 4), this suggests that the averaging was done over the US node and not an N-to-N all-reduce (a detailed analysis of how averaging affects bandwidths is discussed in Section 6). Thus, the limiting factor was the US-ASIA connection at 130 Mb/s rather than the 80 Mb/s from EU-ASIA. The same trend continues with the C-4 run, which adds AUS as a continent with one additional VM. As we know from the transatlantic experiments **(B)** that an additional continent has a detrimental effect on throughput, which, for the four continents experiment, C-4, results in a 9% slower throughput for CV and 36% slower for NLP compared to the A-4 runs (Figure 7a). Again, the US VM is used as an averaging intermediary with a peak egress of 365 Mb/s, while the other continents are between 318 and 330 Mb/s. When comparing the two continents (B-4) versus four continents (C-4) experiments, one GPU on each continent (C-4) is slower by 6% for CV and 20% for NLP compared to two GPUs on two continents (B-4). This reinforces that local hardware should be preferred whenever possible. However, we are always faster than the baseline (A-1), starting from 4 GPUs in both the transatlantic and intercontinental settings. While these experiments were specifically



(a) Throughput (b) Granularity

Figure 10: Multi-cloud performance for CV and NLP.

designed to be a worst-case scenario, what about a more balanced GPU distribution with at least two GPUs in each region?

When comparing the C-6 experiment with two GPUs in three continents to the local A-6 experiments, the throughput slowdown is almost identical (CV 7%, NLP 35%) as with C-4 (CV 9%, NLP 36%) to A-4. Scaling further to two GPUs in four continents, C-8 is slightly slower at NLP (41%) compared to C-4 (36%) to their respective local runs (A-8 and A-4), due to the decreasing granularity of 0.4 (Figure 9b). The small granularity removes the additional gain of four more GPUs since the task is no longer suitable for distributed training. However, as the CV task is still at a granularity of 3.33 on C-8, it reaches a speedup of 3.02x, only 7% slower than the fully local A-8 experiment. The peak egress of 678 Mb/s was also reached on one US VM, while the remaining VMs were between 450 and 550 Mb/s. These observations show that adding another continent does not significantly reduce throughput when training on three continents with at least two VMs.

In summary, while local compute is the best choice for maximum throughput, for high granularity tasks like CV, even distributing VMs over four continents only slows down performance by 7%. However, intercontinental training leads to a significant penalty on a task with lower granularity, like NLP, resulting in a performance drop of 41% (C-8) compared to the fully local experiment (A-8). Finally, each additional region introduces a constant penalty that is not amortized by adding local hardware, which should be considered when running geo-distributed training setups.

5 MULTI-CLOUD PERFORMANCE

Using multiple cloud providers makes sense if we want to use resources cost-effectively and have additional reliability. In our scenario, we are interested in what throughput per $ can be expected and if any barriers prevent multi-cloud training. However, one can also consider the data center's carbon footprint, which can change depending on the season and time of day [5].

From \ To	GC	AWS	Azure
GC	6.35	1.52	0.45
AWS	1.81	4.87	
Azure	0.47		7.63

(a) Single stream TCP throughput in Gb/s.

From \ To	GC	AWS	Azure
GC	0.71	15.3	51.22
AWS	13.85	0.15	
Azure	49.80		1.56

(b) ICMP Latency in ms.

Table 5: Average multi-cloud throughput and latency.

We have compiled the current prices for spot and on-demand instances for T4 GPUs with 8 CPU cores and the egress costs for three well-known cloud providers, GC [4], AWS [1], and Azure [8] (Table 1). There are two different pricing concepts. On the one hand, there are GC and Azure, which offer relatively cheap instances, with 69% and 73% savings over on-demand pricing, respectively, and relatively expensive egress charges between continents of up to $0.15/GB. On the other hand, there is AWS, where the spot instance is only 51% cheaper than the on-demand instance and more than twice as expensive as GC or Azure. However, the egress fees here are much cheaper at only $0.02/GB. Because of the additional offerings around compute, such as networking, identity and cost management, and tooling, it is not easy to fairly compare cloud providers. Therefore, we will limit ourselves to network and VM costs.

With the multi-cloud experiments from this section, we want to evaluate the following scenarios: First, partially switching from one provider to another without stopping the training. Second, scaling resources in the same region when one of the cloud providers is already at capacity for spot-priced VMs or the current price is too high [23]. We know from Section 4 that scaling resources in the same location can significantly improve performance, which may only be possible using additional cloud providers.

Experimental design. To enable a fair comparison between the cloud providers, we rented hardware most similar to each other in the same region. We used each provider's default settings and only changed hardware specs. For GC, it is the same instance as in Section 4. At AWS, it is a `g4dn.2xlarge` with eight cores and 32 GB in the `us-west-2c` region. Unfortunately, we had to make two compromises with Azure. There was only the combination of four cores and 30 GB RAM (`NC4as_T4_v3`), and there were no T4 GPU resources available in the `us-west`, so we had to fall back to `us-south-2`.

The network profiling between all cloud providers in Table 5 shows that their intra-cloud connectivity is comparably fast with 6.4, 4.9, and 7.6 Gb/s for GC, AWS, and Azure, respectively. All connections are mostly symmetric, with inter-cloud connectivity between GC and AWS providing up to 1.8 Gb/s and a ping of 15.3ms, indicating that while they are likely not in the same data center, they are close to each other and connected to the same Internet exchange point. However, connectivity to Azure could be better since it operates in a different zone, with a bandwidth of 0.5 Gb/s and a ping of 51ms.

Our experimental setup consists of four GPUs with equal contributions from each cloud provider. D-1 is the baseline with four GPUs at GC, D-2 with two GPUs each at GC and AWS, and D-3 with two GPUs at GC and Azure. We compare moving two VMs to a different cloud provider to see the impact on cost and throughput.

(1) No inter-cloud throughput penalty. Figure 10 shows the throughput and granularity of each multi-cloud experiment. CV and NLP runs have essentially identical throughput regardless of the combination of cloud providers. Only the D-3 experiments show a very slight slowdown in communication time, reflected in the lower granularity score (Figure 10b) of 12.72 in CV and 1.99 in NLP compared to the D-1 baseline scores of 14.48 and 2.73, respectively. Actual throughput was between 1-2% slower than the baseline, which is negligible and only related to the slightly worse connection to the Azure data center. These results confirm our observation from Section 4 that network connectivity determines scalability, and one can easily train in a multi-cloud scenario.

(2) External egress costs can overshadow VM costs. One drawback to training in multiple regions or zones is that egress traffic can incur additional costs depending on the cloud provider. We have summarized the cost of egress traffic within a zone (intra-zone), between zones in each region (inter-zone), and between continents in Table 1. Notably, any traffic to Oceania (Australia, New Zealand, and others, abbreviated as OCE) generates the highest cost of $0.15/GB for GC. We have broken down the costs for the multi-cloud experiment

(a) Intra- and inter-zone in the US region (D-2/3).

(b) Intercontinental in the US, EU, ASIA and AUS (C-8).

Figure 11: Costs breakdown for D-2/3 and C-8 experiments.

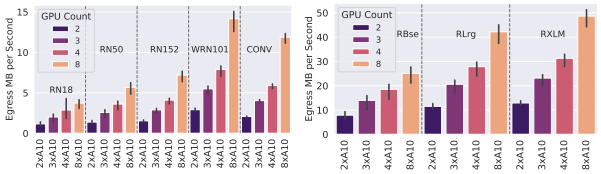
in Figure 11a on an hourly per-VM basis. With only four peers in the D-1/2/3 experiments, we have an N-to-N communication, i.e., each peer sends its gradients to every other peer. This means that $\frac{1}{3}$ of the egress was internal to the partner VM in the same cloud, and the remaining $\frac{2}{3}$ went to the remaining two peers in the other cloud.

First, loading data from Backblaze costs $0.01/GB from anywhere in the world, which gives us a rate of $0.144/h for the CV and $0.083/h for the NLP experiments. Even when CV throughput is less than half of the NLP model (Figure 10a), images are much larger than text, resulting in a higher data rate. While this is close to the spot instance costs of GC ($0.18/h) and Azure ($0.134/h), these are one-time costs until the entire dataset is downloaded and retrieved from the disk cache, assuming large enough local storage. A more detailed comparison of cloud provider storage offerings is beyond our scope, but current prices range from $0.02/GB to $0.14/GB in various GC regions, making our setting (B2) competitive.

Second, the external egress costs for the NLP experiments are very high compared to the other costs. They are 2.2x higher than the spot instance for GC and 5.7x higher for Azure, as the traffic costs in the US zone are $0.01/GB and $0.02/GB, respectively. The Azure cost is even higher ($0.763/h) than the on-demand instance price of $0.489/h. The CV experiments are much less affected due to the smaller model size, but Azure still manages to almost match its spot instance price of $0.134/h with the external egress cost of $0.115/h.

Finally, the total compute cost, including egress and data loading in this multi-cloud constellation, is the sum of all the cloud providers' prices times the number of VMs used. For the CV experiments, GC, AWS, and Azure cost $0.762/h, $1.192/h, and $0.363/h, respectively, making the combination of GC with Azure 42% cheaper than GC with AWS. For the NLP experiments, GC, AWS, and Azure cost $0.835/h, $1.05/h, and $0.973/h, respectively, and GC combined with Azure is better than GC with AWS by a smaller margin of 3.9%. However, the intercontinental network egress prices for both GC and Azure are up to 15 times higher than the inter-zone prices, so what about the cost-effectiveness compared to geo-distributed experiments?

(3) Geo-distributed egress can incur most of the cost. To illustrate the cost of intercontinental training, we use our C-8 experiment with two VMs in four continents from Section 4 to plug the cost for each cloud provider. The egress costs are calculated slightly differently than in the D-2 and D-3 experiments because four groups of two VMs average locally and then distribute the gradients across the other groups. This results in $\frac{8}{20}$ internal egress calls (two calls between each group), $\frac{6}{20}$ intercontinental egress calls (two calls between three regions), and $\frac{6}{20}$ AUS egress calls (three regions share their gradients with AUS and vice versa).



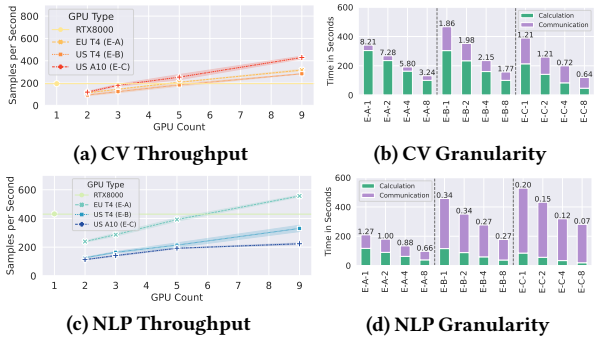
(a) CV (b) NLP

Figure 12: Baseline egress rate on 2-8 A10 GPUs.

Figure 11b shows the resulting egress traffic cost per VM. The high cost between continents scales to a multiple of the remaining cost for CV and NLP with GC and Azure. For NLP, the external egress cost for GC is $4.329/h, more than 90% of the total cost per VM ($4.804/h). Even with Azure having a more moderate rate of $0.02/GB for intercontinental communication and only $0.08/GB for OCE traffic, it still results in $1.882/h external egress cost ($2.101/h total). This is in contrast to AWS, which has a cap of $0.02/GB to any location, resulting in the best total cost of $1.376/h per VM. The relatively high AWS instance cost compares favorably to the other cloud providers regarding geo-distributed training. Keeping egress traffic in mind when deciding to scale to other continents is essential, as it can be the most significant part of the total cost. This raises another question: If egress traffic matters so much, how does model size affect it?

(4) Small models have lower egress rates than larger models. Model size affects two parts of the distributed training time. First, larger models tend to have slower averaging rates, but more data movement costs due to their size. However, larger models are also averaged less frequently because they take longer to perform a step. To analyze this, we review the experiments in Section 3, where we evaluate different model sizes and GPUs counts. Figure 12 shows the average egress rate over each experiment's runtime for both CV and NLP from two to eight A10 GPUs. The trend is clear: the smaller the model, the lower the egress rate for all GPUs (e.g., RN18 vs. RN50). This is surprising, as the "square-cube" law [36] states that with a decrease in parameters, the calculation time will decrease quadratically while the communication time decreases linearly. This means that with a sufficiently small model, most of the training will consist of communication time, and the egress rate would increase, as it is defined through $\frac{\text{parameter count}}{\text{calculation time}}$. However, we find that even with our smallest model, RN18, with 11.7M parameters and eight A10 GPUs, we are still not at the point where the communication time takes up most of the time.

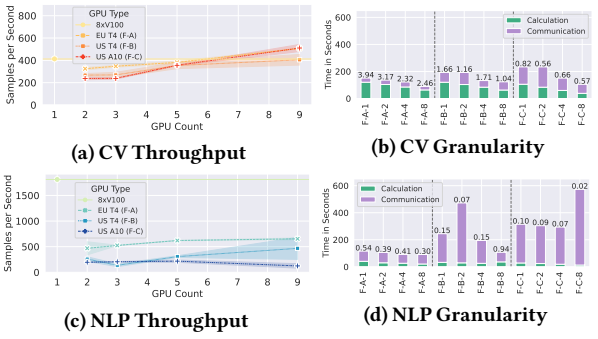
In summary, multi-cloud training is generally possible and can be cost-effective when keeping the egress costs and granularity in mind. Regardless of the cloud provider, staying in the same region is preferred, with the US having the most favorable egress price offers. A significant portion of the cost may be hidden in egress costs, accounting for more than 90% of the total cost in our NLP experiments in GC and Azure. Based on the additional egress costs alone, renting on-demand hardware may be more advantageous than using spot instances between different regions. CV training is generally more calculation- than communication-heavy, resulting in slightly higher data-loading but fewer egress costs. However, from our experiments, this is a favorable trade-off because data-loading is much cheaper than egress costs.



(a) CV Throughput (b) CV Granularity

(c) NLP Throughput (d) NLP Granularity

Figure 13: Hybrid-cloud experiments for the (E) setting.



(a) CV Throughput (b) CV Granularity

(c) NLP Throughput (d) NLP Granularity

Figure 14: Hybrid-cloud experiments for the (F) setting.

6 HYBRID-CLOUD PERFORMANCE

From \ To	EU T4	US T4	US A10
RTX8000	0.45	0.06	0.05
DGX-2 (8xV100)	0.55	0.08	0.07

(a) Single stream TCP throughput in Gb/s.

From \ To	EU T4	US T4	US A10
RTX8000	16.73	150.80	159.05
DGX-2 (8xV100)	16.19	150.27	158.54

(b) ICMP Latency in ms.

Table 6: Average hybrid-cloud throughput and latency.

Can augmenting on-premise hardware with cloud resources be worthwhile to speed up DL training? In this section, we examine two settings: **(E)**, where a consumer-grade GPU, the RTX8000, is deployed on-site, and **(F)**, where a server-grade node, the DGX-2 (8xV100), is deployed on-site. We vary the extra resources, between one to eight T4 EU ({E,F}-A), T4 US ({E,F}-B) and A10 US ({E,F}-C) GPUs.

Experimental design. In both settings, we want to investigate how to extend local hardware with cloud resources and when this leads to better throughput. The cloud resources, in this case, are the same US/EU GC T4 instances as in Section 4 and the US LambdaLabs A10 GPUs from Section 3. We double the number of cloud VMs with each increment, starting with one additional GPU (i.e., E-A-1) until we have eight additional cloud VMs (i.e., E-A-8). This allows us to compare the same hardware in the EU and the US, and slightly weaker, local hardware (EU T4) and better, but more distant hardware (US A10).

Both the **(E)** and **(F)** setups share the network uplink between 450 and 550 Mb/s to the EU datacenter in Belgium, as they are located in the same building in Europe (Table 6). However, as this is not a Google-owned datacenter, the traffic is partly going over the public internet, which results in a lower bandwidth of 50 and 80 Mb/s to the US-based VMs compared to 210 Mb/s between the US and EU GC datacenters (Table 4a).

(E) Consumer-grade setting. The results follow the same trend as in Section 4. The CV task has a higher granularity of 8.21 with 2 GPUs at E-A-1 than NLP (1.27) (Figures 13b and 13d), and scales regardless of the location of the cloud resources (Figure 13a). We almost match the baseline throughput of 195 SPS at 5 GPUs in all settings for CV (E-A-4, E-B-4, E-C-4). The best throughput was reached at E-C-8 with the US A10 GPUs with 429 SPS. For NLP, only the E-A-8 experiment beats the baseline with a speedup of 1.29x and 556 SPS due to the low granularity and the intercontinental base penalty for the US experiments.

Model \ Setup	RTX8000	E-A-8	E-B-8	E-C-8	8xT4	8xA10
CONV	194.8	316.8	283.5	429.3	261.9	620.6
RXLM	431.8	556.7	330.6	223.7	575.1	1059.9

Table 7: Hybrid- vs. cloud-only throughput for the (E) setting.

However, is combining on-premise and remote cloud resources better than using the cloud without paying the intercontinental bandwidth tax? To analyze this, we compare the **(E)** experiments with the 8xA10 experiment from Section 3 and 8xT4 experiment from Section 4 in Table 7. First, the 8xA10 experiments are the fastest for both CV and NLP, which removes the respective hybrid-cloud combination from contention (E-C-8). Second, the 8xT4 experiments for NLP are faster than any other hybrid-cloud setup, making the cloud-only solution favorable. Finally, while we always beat the baseline 8xT4 CV throughput (261.9 SPS), but in the case of E-B-8 (283.5 SPS), just barely. The throughput of E-A-8 (316.8 SPS) makes the hybrid-cloud setup the most favorable in terms of relative GPU scaling (32.5 SPS per GPU), but it does not come close to the best cloud-only throughput of 8xA10 with 620.6 SPS.

Summarizing, the cloud-only experiments are the fastest overall due to their single-GPU throughput and locality. Adding cloud resources to on-premise hardware leads to a high communication time, which is not compensated by the additional processing speed of the GPUs. Proximity to the on-premise hardware is essential, as the more local cloud resources (E-A-8) consistently resulted in a better throughput than the same remote cloud resources (E-B-8).

(F) Server-grade setting. The baseline throughput is significantly higher compared to the RTX8000, with a much more powerful 8xV100 DGX node to 413 SPS for CV and 1811 SPS for NLP (Figures 14a and 14c) via PyTorch data parallelism [25]. This increases the penalties from Section 3, leading to the only speedup from baseline for CV in experiments F-A-8 (507 SPS) and F-C-8 (510 SPS). This is surprising, as the older T4 GPUs in the EU perform similarly to the much newer A10 GPUs in the US, showcasing the trade-off between slower, local compute and faster, remote compute. The granularity of 2.46 for F-A-8 shows that there is enough calculation time to distribute, while the F-C-8 experiments spend ≈ 62% of the total training time on communication with a granularity of 0.57 (Figure 14b). The NLP experiments never reach the baseline throughput of the 8xV100 due to using most of the time for communication. The NLP F-B and F-C experiments mainly consist of communication (Figure 14d) with a granularity of up to 0.02, which results in a nonlinear, unstable training time due to the minimum matchmaking time issue **(2)** from Section 3.

In summary, the hybrid-cloud experiments conclude that while on-premise hardware can be augmented with cloud resources, it will likely be cost-efficient if all resources are on the same continent. Using only cloud resources is more advantageous if the on-premises hardware is not co-located.

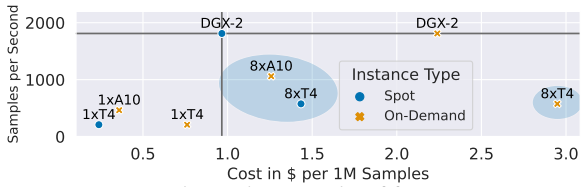


Figure 15: Cost to throughput tradeoff for RoBERTaXLM at different instance types. Our training setups (circled), that are due the low granularity of the NLP model, neither cheaper, nor faster than the centralized offering (DGX-2).

7 FURTHER INSIGHTS

Communication time can decrease with more peers. Let us compare the granularity of the experiments for E-B (Figure 13b), which uses T4 GPUs in the US as an additional cloud resource. *Both* the computation and communication time decrease with the number of GPUs, even increasing the granularity from 1.98 at E-B-2 to 2.15 at E-B-4. This is surprising since, usually, with more peers, the communication time should increase, and the US-EU communication bottleneck should slow us down to the same extent as the E-B-1 experiment. This reduction is a Hivemind-specific anomaly, as it uses a single TCP stream per peer. With TCP, there needs to be an acknowledgment (ACK) of each packet by the receiving peer, which is impacted by the connection's latency. In our high latency network between continents, the round trip time (RTT) of 300-318ms limits the maximum bandwidth a single TCP stream to 50-80 Mb/s. However, a way to improve link utilization is to use multiple streams, one for each peer, which we encounter in experiments E-(B|C)-2,4,8. To verify the potential gains, we perform a microbenchmark of the multi-stream bandwidth from the RTX8000 to the EU and US data centers.Although there is wide variation, likely due to network utilization, with 80 clients, we achieve a maximum bandwidth of 6 Gb/s within the EU and up to 4 Gb/s to the US. While larger peer groups and, consequently, larger models benefit from multi-peer communication by default and do not see significant changes in communication time, small models in unevenly distributed VMs setups can be disproportionately affected. The same trend can be observed in all high latency experiments (i.e., between the EU and the US), e.g., E-B, E-C for CV and NLP (Figures 13b and 13d, and F-B and F-C for CV (Figure 14b). In summary, uneven distribution of computational resources in high-latency networks (e.g., intercontinental) can reduce communication time with Hivemind due to more parallelism, lessening the impact of low bandwidth for a single data stream.

Cost analysis. The DGX-2 (8xV100) node from Section 6 represents server-grade hardware that could be used to train models. However, how does it compare in throughput per $ to all of our distributed cloud experiments? The Figure 1 (CV) and Figure 15 (NLP) show the complete cost analysis of the DGX-2, the 8xT4 experiments, and the 8xA10 experiments for spot and on-demand pricing. We use the internal egress costs from Figure 11a as a reference for the 8xT4 setup. For simplicity, we compare the spot pricing without interruptions, as we assume that a new VM can be spun up fast enough not to affect the training throughput in the long run. We mark the centralized baseline (DGX-2) cost per 1M samples and the throughput in samples per second with a horizontal and vertical line. This means that we are cheaper to the left to the vertical line, and above the horizontal line, we are faster (and vice versa). We circle the new value propositions that we enable in both figures. Our hardware setups have additional key characteristics: They are resilient by default to interruptions due to running in a decentralized fashion and they enable the combination of more GPUs than cloud providers offer in a single node. Currently, common hardware configurations (DGX) allow up to eight GPUs connected via NVLink, and with older hardware, only up to 4xT4s connected via PCIe at 10 GB/s between GPUs (with GC). We were able to combine eight single GPU nodes from GC and LambdaLabs to create competing performance and price setups without dedicated GPU interconnects.

A spot DGX-2 costs at the time of writing $6.30/h ($14.60/h on-demand) in GC US, which makes it the best value proposition for the low granularity NLP task. It is followed by the 8xA10, which are 41% slower and 30% more expensive than the DGX-2 (Figure 15). The 8xT4 experiments are even more expensive, as the internal egress costs take up more than half of the costs, making them the worst value proposition. However, for CV, we manage to provide two new offerings: First, the 8xA10, which is both 50% faster and 49% cheaper than the DGX-2, and 8xT4, which is 58% cheaper than DGX-2, while being 37% slower (Figure 1). The CV model can be scaled more easily due to its initially high granularity, which makes the very competitive offering of $0.6/h per A10 from LambdaLabs an excellent value proposition. However, while we only evaluated eight T4 GPUs for our GC-based experiments, with a granularity of 5.19 (CV A-8 in Figure 7b), there is ample space to scale even further. It is important to note that LambdaLabs does not charge for any data egress, but GC does with $0.01/GB, and the 8xT4 experiment is still cheaper. While LambdaLabs is often at capacity, Google Cloud positions itself as a hyperscaler with the advantage of rarely being at max occupancy.

We also evaluated the performance of the 4xT4 PyTorch DDP [25] for CV with the best available multi-T4 node on GC (4xT4). The NLP experiments ran OOM. Since the DDP 4xT4 runs on a single node, it causes no interconnect costs and is priced at $0.96 per 1M samples at spot pricing, while our 8xT4 setup costs $1.77 per 1M samples (84% more expensive). However, the 8xT4 setup has a higher throughput of 262 SPS (26% faster) compared to the 4xT4 node (207 SPS). This higher speed is not available at the price point of the 4xT4 node. Moreover, the 8xT4 setup has the potential for further scaling, which we discussed in detail in Section 4.

In summary, the lower spot prices for older GPUs allow us to train models more cost-efficiently when task granularity allows it and get more value per $ when training on the 8xT4 or 8xA10 compared to an DGX-2 node. Combining multiple nodes with single GPUs with lower bandwidths enables scaling that was previously impossible to achieve without resorting to much more powerful GPUs. Distributed spot instance pricing opens up a new value proposition compared to on-demand offerings that can even compete with the competitive pricing of smaller cloud providers.

Spot VM Interruption Frequency. While we used low spot prices as a cost-saving argument in our experiments, we did not elaborate on the most significant drawback - the possibility of being terminated by the cloud provider at any time. There is already some research on how different cloud providers track the interruption frequency and can be used for varying workloads to achieve a positive $-per-throughput effect [23, 42, 43].

Interruption affects three aspects: First, the interruption frequency is defined by AWS as the number of VMs terminated in the last 30

days, which is between 5 and 20% [2]. This value was not representative during our experiments with any cloud provider, as we noticed that it is highly dependent on the time of day of the zone.

Second, the time needed to setup a VM until training starts. The startup time of a VM depends on the cloud provider (e.g., a preconfigured image) and the one's technology stack (e.g., Docker, Kubernetes, Ansible). In our experience, VM startup time ranges between seconds to minutes with manual deployment taking up to 10 minutes. Although startup time can be improved, model training typically takes multiple hours or days, making it a less impactful optimization.

Third, the time required for the new peer to synchronize the training state with other peers. In our experience, this took at worst two hivemind epochs due to the averaging starting before synchronization is finished. While it is possible to create a hivemind epoch that is short enough to prevent new peers from joining, this only happens with a low enough granularity where scaling is not beneficial anymore as we are mostly communication bound.

Finally, while the VM setup and synchronization of the training state take time, the interruption frequency significantly affects the final throughput. We faced difficulties acquiring even a single spot VM during our GC experiments during daylight hours. This highlights the need for systems like SkyPilot [43], which utilizes automation to deploy spot instances across various clouds and zones. In our case, the interruption frequency can be used as a penalty on the training throughput, i.e., a 5% interruption frequency over the entire training time means roughly a 5% slower training.

8 LESSONS LEARNED

We find it important to summarize our findings more generically to provide guidance for DL practitioners that want to perform distributed spot training. These lessons are based on the Sections 3 to 6.

Small model training still scales. We have shown that models between 12M-560M parameters can be trained in a decentralized, distributed fashion achieving a speedup of up to 4.37x on eight Ampere-GPUs. The limiting factor as to when a model is suitable for (geo-)distributed training is the target batch size which all peers need to accumulate until synchronization happens. We found a TBS of 32K suitable to not only train in a single zone, but even see a speedup when using VMs in four different continents. As long as the optimizer can handle big-batch training and the dataset is big enough to accommodate large batches, the remaining issue to find the base granularity of the model to decide how to scale it cost-effectively. Finally, we found that small models induce less traffic over larger models over time, even at a much higher averaging rate, making them better suited for cost-efficient training than large models.

Egress costs can take up most of the total cost. Egress pricing for the NLP experiments overtook the spot and the on-demand costs of T4 GPUs when training on four continents or even in two zones. For example, RoBERTaXLM's high throughput and parameter count require more data to be sent between peers during averaging due to smaller granularity. Under the current pricing models, AWS has the best value for geo-distributed training, while GC and Azure are best at training in a single zone. The biggest cost-saving potential lies in cloud providers that do not charge for egress at all, like LambdaLabs.

Granularity is important to evaluate scalability. We found that the ratio between calculation and communication time, granularity, is the most important metric to track when deciding on distributed training suitability. It enables us to compare the scalability potential between different models on the same hardware due to summarizing their model size and throughput ratio. Additionally, it gives a value to the cost-efficiency: With a granularity of exactly 1, the potential speedup when doubling the number of VMs is, at best, 1.33x due to halving the calculation time. However, with a granularity of 10, the speedup with double the VMs is, at best, 1.83x due to the communication time playing a less significant role. With this, we can estimate training performance with additional resources.

Geo-distributed multi-cloud training is possible and is cost-efficient. Even with the current teething pains of Hivemind, we got a speedup in all of our experimental setups of intra-zone, transatlantic, and intercontinental settings as long as the granularity of the task permitted it. Using older and cheaper Tesla GPUs at spot pricing is not only more cost-efficient than the DGX-2 offering, but even trumps the competitive pricing model of LambdaLabs, all while including egress costs. Our network profiling showed that the current training limitations are not primarily the bandwidth but rather the intercontinental latency and the task's granularity. If the granularity is already low at high bandwidth, it can only worsen when used in a high latency, low bandwidth network. When considering both, estimating the potential cost-savings of investing in a multi-/hybrid-cloud scenario is possible.

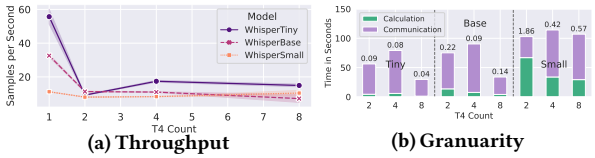
9 RELATED WORK

Decentralized deep learning. Training with unreliable peers has been studied in a collaborative setting, resulting in the Distributed Deep Learning in Open Collaborations (DeDLOC) [16] algorithm, on which the Hivemind framework [39] is based. It can interpolate between traditional distributed DL algorithms like parameter servers [24], decentralized SGD [26], or All-Reduce SGD [38]. We used the Hivemind framework for all of our experiments, as it provided the base for training on spot instances in high latency, low bandwidth networks.

SWARM [36] applies both previous techniques and adds model parallelism to the mix by creating pipelines between nodes and rebalancing them in case of failures. The authors find a crucial insight in the "square-cube" law, which argues for better training scalability with larger model sizes; as the size increases linearly, so does the communication time, while the calculation time increases quadratically. We add to that by analyzing distributed training for smaller model sizes that pose different trade-offs. We show that while the square-cube law still holds for increasing model sizes, under consideration of granularity, we can still train small models.

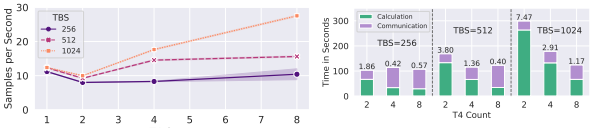
Decentralized deep learning on heterogeneous hardware with slow interconnects can benefit the training of foundation models. To achieve this, model and pipeline parallelism can be used in addition to data-parallel training [45]. This is a complementary work to ours, since we target smaller models and weaker hardware.

Deep learning on spot instances. DeepSpotCloud [22] is a system that uses the AWS API to automatically migrate a DL task with checkpointing whenever the spot instance is terminated. The authors note that the volatility of GPU instance pricing and interruptions have a unique pattern compared to non-accelerated VMs, and solve this by using intercontinental provisioning. We noticed the same trends of high interruption ratios in our experiments. However, we have shown that geo-distributed training is possible until



(a) Throughput (b) Granularity

Figure 16: ASR model performance with TBS=256.



(a) Throughput (b) Granularity

Figure 17: WhisperSmall performance with varying TBS.

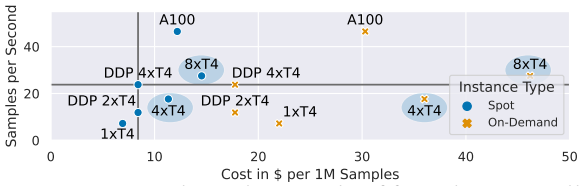


Figure 18: Cost to throughput tradeoff for WhisperSmall at TBS=1024 with different instance types. Our training setups (circled) provide mixed result of being slightly faster and more expensive than comparable, centralized DDP offering.

granularity permits it, which poses a possibility for ever-migrating training between continents without checkpointing.

Amazon Sagemaker [13] is an AWS service that allows to perform ML under budget constraints. For training, it supports spot VM migration until a cost threshold is reached by checkpointing the progress. However, it lacks the option of training on multiple spot VMs. It can do either spot instance training on DGX-like nodes or combine multiple on-demand nodes with PyTorch DDP (or similar), but not both. This eliminates the potential of accelerating the training process with more GPUs that do not fit a single spot-provisioned hypervisor.

The analysis by Yang et al. [42] investigates maximizing a target accuracy from a spot pricing versus time perspective. Linear programming was used to decide how to provision the VMs with different cost-utility trade-offs. While this shows the potential of utilizing multiple clouds and continents for non-distributed tasks, we evaluated the distributed spot training problem from the throughput, cost, and model size perspective on different hardware setups. By including our insights, their technique for scheduling on spot instances could be adapted to optimize the total throughput of all peers.

Skypilot [43] is a broker system where users can submit their hardware requirements, and it tries to provision the necessary resources on any supported cloud. It features a preemption analysis that counts the number of interruptions in a zone and can decide to migrate whenever they cross a certain threshold. We have shown that multi-, hybrid-cloud, and geo-distributed training is possible, and by combining our insights, it would open up auto-migrated, decentralized DL training for the best spot prices in the world.

10 CONCLUSION

This paper analyzes multi- and hybrid-cloud training in a decentralized fashion on spot instances. We define the lower bounds of model sizes that can be scaled cost-efficiently using the granularity metric to estimate their suitability for distributed training in low-bandwidth, high-latency situations. We show that training on multiple cloud providers and four continents still scales with additional compute resources. Alternatively to the current use of spot instances in DL, we show the potential of using spot instances in a distributed, decentralized way by being more cost-efficient with eight T4 instances over a DGX-2 from the same cloud provider while paying additional egress costs. Finally, we provide an intuition about where costs in such a training scenario come from and how different model sizes from CV and NLP affect throughput and costs. Our work empowers practitioners to utilize spot-priced instances for distributed deep learning with relatively small models. Our insights show some potential that can further improve distributed training performance, such as optimizers with higher minibatch sizes and improvements regarding the communication time with, e.g., better compression.

11 APPENDIX: ASR CASE STUDY

We perform a case study on Automatic Speech Recognition (ASR) to showcase spot training on weaker GPUs. Whisper [33] is a state-of-the-art ASR model trained on 680,000 hours of labeled data to transcribe audio. It features different sizes, from 37.8M to 1.5B parameters, and was trained with a minibatch size of 256. We use the Common-voice [10] dataset, preprocessed to Log-Mel spectrograms. In our distributed experiments, we start with a TBS of 256 and increase to 512 and 1024 to combat potential granularity issues. Due to memory constraints, only three model sizes (Tiny, Base, Small) were trainable on the T4 GPU. Unfortunately, the original TBS of 256 was not large enough to train the relatively small models due to their small granularity (Figure 16). This leads to no performance benefits compared to running on a single T4 GPU. In some cases, the low throughput and non-deterministic communication time (8xT4 Tiny, 8xT4 Base) was the result of some peers not being able to join due to never being able to sync the current model state (cf. Section 7). The only model showing scaling potential is WhisperSmall, with a granularity of 1.8 with 2xT4. However, when scaling the target batch size to 512 and 1024, we see some benefit over the single GPU runs for the WhisperSmall model (Figure 17). By effectively increasing the amount of computation by the factors of 2 and 4, we can generate a speedup of 1.27× and 2.2× with 8xT4's for the TBS 512 and 1024, respectively. When compared to other hardware setups, our A100 80GB GPU and the best multi-T4 GPU on GC (4xT4) with Pytorch DDP (Figure 18) have almost double the throughput at 46 SPS and are slightly slower at 24 SPS, respectively, compared to our 8xT4 setup which runs at 28 SPS. This outcome is not surprising due to the generational leap in architecture for the A100 and the slower interconnect with our 8xT4 experiments compared to a single 4xT4 node (see Section 3 for a detailed throughput analysis). The proposed cost-throughput ratio is mixed: the A100 is at $12.19/1M samples, the DDP 4xT4 is at $8.41/1M, and our 8xT4 is at $14.53/1M. Our proposed setup is slightly more expensive than the A100, and it will not scale beyond eight T4 GPUs due a granularity at 1.17, leaving the A100 as the fastest and the DDP 4xT4 setup as the cheaper but slower alternative. Despite these results, our proposed setup has several benefits, including resilience for spot interruptions, interruption-free migration to the lowest cloud prices, and the possibility to scale the GPU count up as long as granularity permits it.

ACKNOWLEDGMENTS

This work is funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 392214008.

REFERENCES

[1] 2023. *Amazon AWS.* Accessed: 19 May 2023, aws.amazon.com.

[2] 2023. Amazon AWS Spot Pricing. https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/. Accessed: 2023-09-27.

[3] 2023. Backblaze. https://backblaze.com/. Accessed: 2023-10-05.

[4] 2023. *Google Cloud.* Accessed: 19 May 2023, cloud.google.com.

[5] 2023. Google Cloud Region Picker. https://cloud.withgoogle.com/region-picker/. Accessed: 2023-10-05.

[6] 2023. Hivemind GAC Issue. https://github.com/learning-at-home/hivemind/issues/566. Accessed: 2023-10-05.

[7] 2023. *LambdaLabs.* Accessed: 19 May 2023, lambdalabs.com.

[8] 2023. *Microsoft Azure.* Accessed: 19 May 2023, portal.azure.com.

[9] Alex Aizman, Gavin Maltby, and Thomas Breuel. 2019. High Performance I/O For Large Scale Deep Learning. In *2019 IEEE International Conference on Big Data (Big Data)*. 5965–5967. https://doi.org/10.1109/BigData47090.2019.9005703

[10] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M Tyers, and Gregor Weber. 2019. Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670* (2019).

[11] Alexander Borzunov, Max Ryabinin, Tim Dettmers, Quentin Lhoest, Lucile Saulnier, Michael Diskin, and Yacine Jernite. 2022. Training Transformers Together. In *NeurIPS 2021 Competitions and Demonstrations Track*. PMLR, 335–342.

[12] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. arXiv:1911.02116 [cs.CL]

[13] Piali Das, Nikita Ivkin, Tanya Bansal, Laurence Rouesnel, Philip Gautier, Zohar Karnin, Leo Dirac, Lakshmi Ramakrishnan, Andre Perunicic, Iaroslav Shcherbatyi, Wilton Wu, Aida Zolic, Huibin Shen, Amr Ahmed, Fela Winkelmolen, Miroslav Miladinovic, Cedric Archembeau, Alex Tang, Bhaskar Dutt, Patricia Grao, and Kumar Venkateswar. 2020. Amazon SageMaker Autopilot: A White Box AutoML Solution at Scale. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning* (Portland, OR, USA) *(DEEM'20)*. Association for Computing Machinery, New York, NY, USA, Article 2, 7 pages. https://doi.org/10.1145/3399579.3399870

[14] 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*. Ieee, 248–255.

[15] Tim Dettmers. 2016. 8-Bit Approximations for Parallelism in Deep Learning. arXiv:1511.04561 [cs.NE]

[16] Michael Diskin, Alexey Bukhtiyarov, Max Ryabinin, Lucile Saulnier, Anton Sinitsin, Dmitry Popov, Dmitry V Pyrkin, Maxim Kashirin, Alexander Borzunov, Albert Villanova del Moral, et al. 2021. Distributed Deep Learning In Open Collaborations. *Advances in Neural Information Processing Systems* 34 (2021), 7879–7897.

[17] O Elharrouss, Y Akbari, N Almaadeed, and S Al-Maadeed. [n.d.]. Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches. arXiv 2022. *arXiv preprint arXiv:2206.08016* ([n. d.]).

[18] Anne C Elster and Tor A Haugdahl. 2022. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering* 24, 2 (2022), 95–100.

[19] Wikimedia Foundation. 2023. *Wikimedia Downloads.* https://dumps.wikimedia.org

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[21] Kai Hwang. 1992. *Advanced Computer Architecture: Parallelism,Scalability,Programmability* (1st ed.). McGraw-Hill Higher Education.

[22] Kyungyong Lee and Myungjun Son. 2017. DeepSpotCloud: Leveraging Cross-Region GPU Spot Instances for Deep Learning. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. 98–105. https://doi.org/10.1109/CLOUD.2017.21

[23] Sungjae Lee, Jaeil Hwang, and Kyungyong Lee. 2022. SpotLake: Diverse Spot Instance Dataset Archive Service. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*. 242–255. https://doi.org/10.1109/IISWC55918.2022.00029

[24] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 583–598.

[25] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[26] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in neural information processing systems* 30 (2017).

[27] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]

[28] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11976–11986.

[29] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2020. Mlperf training benchmark. *Proceedings of Machine Learning and Systems* 2 (2020), 336–349.

[30] Petar Maymounkov and David Mazieres. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers*. Springer, 53–65.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[32] Gustavo Portella, Genaina N Rodrigues, Eduardo Nakano, and Alba CMA Melo. 2019. Statistical analysis of Amazon EC2 cloud pricing models. *Concurrency and Computation: Practice and Experience* 31, 18 (2019), e4451.

[33] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, and Ilya Sutskever. 2023. Robust Speech Recognition via Large-Scale Weak Supervision. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.), Vol. 202. PMLR, 28492–28518. https://proceedings.mlr.press/v202/radford23a.html

[34] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.

[35] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. ZeRO-Offload: Democratizing Billion-Scale Model Training. arXiv:2101.06840 [cs.DC]

[36] Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. 2023. SWARM Parallelism: Training Large Models Can Be Surprisingly Communication-Efficient. *arXiv preprint arXiv:2301.11913* (2023).

[37] Max Ryabinin, Eduard Gorbunov, Vsevolod Plokhotnyuk, and Gennady Pekhimenko. 2021. Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices. In *Advances in Neural Information Processing Systems*, Vol. 34. https://proceedings.neurips.cc/paper/2021/file/97275a23ca44226c9964043c8462be96-Paper.pdf

[38] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[39] Learning@home team. 2020. Hivemind: a Library for Decentralized Deep Learning. https://github.com/learning-at-home/hivemind.

[40] Chathurika S. Wickramasinghe, Daniel L. Marino, and Milos Manic. 2021. ResNet Autoencoders for Unsupervised Feature Learning From High-Dimensional Data: Deep Models Resistant to Performance Degradation. *IEEE Access* 9 (2021), 40511–40520. https://doi.org/10.1109/ACCESS.2021.3064819

[41] Mitchell Wortsman, Tim Dettmers, Luke Zettlemoyer, Ari Morcos, Ali Farhadi, and Ludwig Schmidt. 2023. Stable and low-precision training for large-scale vision-language models. *arXiv preprint arXiv:2304.13013* (2023).

[42] Sheng Yang, Samir Khuller, Sunav Choudhary, Subrata Mitra, and Kanak Mahadik. 2022. Scheduling ML Training on Unreliable Spot Instances. In *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion* (Leicester, United Kingdom) *(UCC '21)*. Association for Computing Machinery, New York, NY, USA, Article 29, 8 pages. https://doi.org/10.1145/3492323.3495594

[43] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, and Ion Stoica. 2023. SkyPilot: An Intercloud Broker for Sky Computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 437–455. https://www.usenix.org/conference/nsdi23/presentation/yang-zongheng

[44] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962* (2019).

[45] Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy S Liang, Christopher Re, and Ce Zhang. 2022. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems* 35 (2022), 25464–25477.

[46] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).

PVLDB Copyright License Form

Proceedings of the VLDB Endowment

Title of Work: How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study

Author(s): Alexander Erben, Ruben Mayer, Hans-Arno Jacobsen

Select either option A or B by checking [X] the appropriate box below:

☐ A: The undersigned certifies that the work identified above (the "Work") has been written in the course of employment by a government agency or other organization whereby, as a result of such employment, no copyright in the Work exists.

☑ B: If and when the Work is accepted by the VLDB Endowment for publication, the undersigned hereby grants to the VLDB Endowment the following:

1. An exclusive, worldwide, irrevocable and royalty-free right to publish the Work first in audio, video, electronic, and/or digital form, and any and all other media and distribution mechanisms now known or later developed;
2. A non-exclusive, worldwide, perpetual, irrevocable and royalty-free license to reproduce, use, publicly perform, modify the formatting of and create derivative works of the Work;
3. The right to license to third parties ("Third Party Publishers") the right to reproduce, use, modify the formatting of, create derivative works of the Work and to publish, publicly perform, distribute and sell copies of the Work as a compilation or collective work, for any commercial or non-commercial purpose, in digital form and all other media and distribution mechanisms now known or later developed; and
4. The right to license to Third Party Publishers the right to reproduce, use, modify the formatting of and create derivative works of excerpts of the Work and to publish, publicly perform, distribute and sell copies of such excerpts incorporated with other works of authorship as a compilation or collective work, for any commercial or non-commercial purpose, in digital form and all other media and distribution mechanisms now known or later developed; provided that such compilation or collective work shall include an acknowledgement of the author's contribution.

The license in sub-paragraph 1. above includes right to (a) publish and distribute the Work under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) License (http://creativecommons.org/licenses/by-nc-nd/4.0/) and (b) obtain a Digital Object Identifier ("DOI") for the Work.

The right in sub-paragraph 3 above includes permitting Third Party Publishers to use the DOI for the Work to enable others to access an abstract of the Work or portions of the Work and purchase digital copies of the Work.

The license and rights granted above do not transfer ownership of the copyright in the Work to the VLDB Endowment. The author(s) reserve(s) all rights in the Work not expressly granted.

This instrument must be signed by at least one author of the Work or, in the case where the Work was commissioned by another person or organization or created as part of the author's duties as an employee, an authorized representative of the commissioning person or organization or of the employer.

I am an author of the Work, an agent of an author of the Work, a representative of the person or organization that commissioned the Work or a representative of the author's employer and I represent and warrant that I have full power and authority to grant the above license of the Work.

Print Name: Alexander Erben

Signature 

Date 15.02.2024 Title (if not author)

Bibliography

[Bro+20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language models are few-shot learners." In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[Cer24] Cerebras. *Wafer-Scale Engine-2*. `https://web.archive.org/web/20240000000000*/https://www.cerebras.net/product-chip/`. 2024.

[Che+21] J. Chen, C. Wolfe, Z. Li, and A. Kyrillidis. *Demon: Improved Neural Network Training with Momentum Decay*. 2021. arXiv: `1910.04952 [cs.LG]`.

[Cho+23] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. "Palm: Scaling language modeling with pathways." In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.

[Cli22] I. P. on Climate Change (IPCC). "Summary for Policymakers." In: *Global Warming of 1.5°C: IPCC Special Report on Impacts of Global Warming of 1.5°C above Pre-industrial Levels in Context of Strengthening Response to Climate Change, Sustainable Development, and Efforts to Eradicate Poverty*. Cambridge University Press, 2022, pp. 1–24.

[DAW22] N. Ding, M. Awan, and S. Williams. "Instruction Roofline: An insightful visual performance model for GPUs." In: *Concurrency and Computation: Practice and Experience* 34.20 (2022), e6591.

[Det+22] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. "GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale." In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 30318–30332.

[Dev+18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018).

[Dis+21] M. Diskin, A. Bukhtiyarov, M. Ryabinin, L. Saulnier, A. Sinitsin, D. Popov, D. V. Pyrkin, M. Kashirin, A. Borzunov, A. Villanova del Moral, et al. "Distributed deep learning in open collaborations." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 7879–7897.

[DZ19] T. Dettmers and L. Zettlemoyer. "Sparse networks from scratch: Faster training without losing performance." In: *arXiv preprint arXiv:1907.04840* (2019).

[EB22] E. Erdil and T. Besiroglu. "Algorithmic progress in computer vision." In: *arXiv preprint arXiv:2212.05153* (2022).

[EMJ23] A. Erben, R. Mayer, and H.-A. Jacobsen. *How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study*. 2023. arXiv: 2306.03163 [cs.LG].

[Erb+22] A. Erben*, R. Mayer, J. Jedele, and H.-A. Jacobsen. "Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines." In: *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD '22. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 1825–1839. ISBN: 9781450392495. DOI: 10.1145/3514221.3517848.

[Goo15] Google. *Gemini*. https://web.archive.org/web/20240000000000*/https://cloud.google.com/tpu/docs/intro-to-tpu. 2015.

[Goo23] Google. *Gemini*. https://web.archive.org/web/20240000000000*/https://deepmind.google/technologies/gemini. 2023.

[Goo24] Google. *Google Data Center Locations And Investment*. https://web.archive.org/web/20240701000000*/https://www.google.com/about/datacenters/locations. 2024.

[Goy+17] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. "Accurate, large minibatch sgd: Training imagenet in 1 hour." In: *arXiv preprint arXiv:1706.02677* (2017).

[GRK17] S. Gray, A. Radford, and D. P. Kingma. "Gpu kernels for block-sparse weights." In: *arXiv preprint arXiv:1711.09224* 3.2 (2017), p. 2.

[Hof+22] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. "Training compute-optimal large language models." In: *arXiv preprint arXiv:2203.15556* (2022).

[Hua+19] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." In: *Advances in neural information processing systems* 32 (2019).

[IBM23] IBM. *A new chip architecture points to faster, more energy-efficient AI*. `https://web.archive.org/web/20240000000000*/https://research.ibm.com/blog/northpole-ibm-ai-chip`. 2023.

[Int23] Intel. *Gaudi3*. `https://web.archive.org/web/20240000000000*/https://www.intel.com/content/www/us/en/newsroom/news/ai-everywhere-core-ultra-5th-gen-xeon-news.html`. 2023.

[Iva+21] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler. "Data movement is all you need: A case study on optimizing transformers." In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 711–732.

[Jia+20] Z. Jiang, L. Wang, X. Xiong, W. Gao, C. Luo, F. Tang, C. Lan, H. Li, and J. Zhan. "HPC AI500: The methodology, tools, roofline performance models, and metrics for benchmarking hpc ai systems." In: *arXiv preprint arXiv:2007.00279* (2020).

[Kak+19] A. Kakaraparthy, A. Venkatesh, A. Phanishayee, and S. Venkataraman. "The case for unifying data loading in machine learning clusters." In: *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. 2019.

[Kan+20] D. Kang, A. Mathur, T. Veeramacheneni, P. Bailis, and M. Zaharia. "Jointly optimizing preprocessing and inference for DNN-based visual analytics." In: *arXiv preprint arXiv:2007.13005* (2020).

[KB14] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems* 25 (2012).

[LH17] I. Loshchilov and F. Hutter. "Decoupled weight decay regularization." In: *arXiv preprint arXiv:1711.05101* (2017).

[Li+20] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, et al. "Pytorch distributed: Experiences on accelerating data parallel training." In: *arXiv preprint arXiv:2006.15704* (2020).

[Lia+17] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent." In: *Advances in neural information processing systems* 30 (2017).

[MET24] META. *META Data Center Locations And Investment*. https://web.archive.org/web/20230701000000*/https://datacenters.atmeta.com/all-locations. 2024.

[Mic23] Microsoft. *Microsoft announces $5 billion investment in compute capacity*. https://web.archive.org/web/20230901000000*/https://news.microsoft.com/en-au/features/microsoft-announces-a5-billion-investment-in-computing-capacity-and-capability-to-help-australia-seize-the-ai-era. 2023.

[Moh+20] J. Mohan, A. Phanishayee, A. Raniwala, and V. Chidambaram. "Analyzing and mitigating data stalls in DNN training." In: *arXiv preprint arXiv:2007.06775* (2020).

[Nar+19] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia. "PipeDream: Generalized pipeline parallelism for DNN training." In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 2019, pp. 1–15.

[Nar+21] D. Narayanan, A. Phanishayee, K. Shi, X. Chen, and M. Zaharia. "Memory-efficient pipeline-parallel dnn training." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 7937–7947.

[Niu+21] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren. "DNNFusion: accelerating deep neural networks execution with advanced operator fusion." In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 2021, pp. 883–898.

[Ofe+14] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, and M. Püschel. "Applying the roofline model." In: *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. 2014, pp. 76–85.

[Ope22] OpenAI. *ChatGPT*. https://web.archive.org/web/20240000000000*/https://openai.com/blog/chatgpt. 2022.

[PyT23] PyTorch. *torch.compile*. https://web.archive.org/web/20240000000000*/https://pytorch.org/tutorials/intermediate/torch_compile_tutorial.html. 2023.

[Ras+20] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He. "DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters." In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 3505–3506. ISBN: 9781450379984. DOI: 10.1145/3394486.3406703.

[Ren+21] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He. "{ZeRO-Offload}: Democratizing {Billion-Scale} model training." In: *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 2021, pp. 551–564.

[RM51] H. Robbins and S. Monro. "A stochastic approximation method." In: *The annals of mathematical statistics* (1951), pp. 400–407.

[Rya+23] M. Ryabinin, T. Dettmers, M. Diskin, and A. Borzunov. "Swarm parallelism: Training large models can be surprisingly communication-efficient." In: *arXiv preprint arXiv:2301.11913* (2023).

[SGM19] E. Strubell, A. Ganesh, and A. McCallum. "Energy and policy considerations for deep learning in NLP." In: *arXiv preprint arXiv:1906.02243* (2019).

[Sho+19] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. "Megatron-lm: Training multi-billion parameter language models using model parallelism." In: *arXiv preprint arXiv:1909.08053* (2019).

[Sil+23] C. Silvano, D. Ielmini, F. Ferrandi, L. Fiorin, S. Curzel, L. Benini, F. Conti, A. Garofalo, C. Zambelli, E. Calore, et al. "A survey on deep learning hardware accelerators for heterogeneous hpc platforms." In: *arXiv preprint arXiv:2306.15552* (2023).

[Tec18] TechPowerUP. *NVIDIA Tesla T4*. `https://www.techpowerup.com/gpu-specs/tesla-t4.c3316`. 2018.

[TKC19] P. Tillet, H.-T. Kung, and D. Cox. "Triton: an intermediate language and compiler for tiled neural network computations." In: *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 2019, pp. 10–19.

[VT16] M. Van Steen and A. S. Tanenbaum. "A brief introduction to distributed systems." In: *Computing* 98 (2016), pp. 967–1009.

[Wie+10] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. "Conductor: orchestrating the clouds." In: *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware*. 2010, pp. 44–48.

[Wik24] Wikipedia contributors. *List of Nvidia graphics processing units*. [Online; accessed 12-February-2024]. 2024.

[Yan+23] Z. Yang, Z. Wu, M. Luo, W.-L. Chiang, R. Bhardwaj, W. Kwon, S. Zhuang, F. S. Luan, G. Mittal, S. Shenker, et al. "{SkyPilot}: An Intercloud Broker for Sky Computing." In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 2023, pp. 437–455.

[You+19] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh. "Large batch optimization for deep learning: Training bert in 76 minutes." In: *arXiv preprint arXiv:1904.00962* (2019).

[Yua+22] B. Yuan, Y. He, J. Davis, T. Zhang, T. Dao, B. Chen, P. S. Liang, C. Re, and C. Zhang. "Decentralized training of foundation models in heterogeneous environments." In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 25464–25477.

[Zha+22] B. Zhang, L. Luo, X. Liu, J. Li, Z. Chen, W. Zhang, X. Wei, Y. Hao, M. Tsang, W. Wang, Y. Liu, H. Li, Y. Badr, J. Park, J. Yang, D. Mudigere, and E. Wen. *DHEN: A Deep and Hierarchical Ensemble Network for Large-Scale Click-Through Rate Prediction*. 2022. arXiv: 2203.11014 [cs.IR].

[Zha+23] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, et al. "Pytorch FSDP: experiences on scaling fully sharded data parallel." In: *arXiv preprint arXiv:2304.11277* (2023).

[Zhu+18] Y. Zhu, F. Chowdhury, H. Fu, A. Moody, K. Mohror, K. Sato, and W. Yu. "Entropy-aware I/O pipelining for large-scale deep learning on HPC systems." In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2018, pp. 145–156.