ML Training with Cloud GPU Shortages: Is Cross-Region the Answer?

Foteini Strati
ETH Zurich

Paul Elvinger
ETH Zurich

Tolga Kerimoglu
ETH Zurich

Ana Klimovic
ETH Zurich

Abstract

The widespread adoption of ML has led to a high demand for GPU hardware and consequently, severe shortages of GPUs in the public cloud. Allocating a sufficient number of GPUs to train or fine-tune today's large ML models in a single cloud region is often difficult. Users can get access to more GPUs if they are willing to run a ML training job using devices across different geographical regions. However, GPU nodes are connected with lower network bandwidth and cloud providers charge extra for data transfers across geographical regions. In this work, we explore when and how it makes sense to leverage GPUs across zones and regions for distributed ML training. We analyze the throughput and cost impact of cross-region training based on the computation and communication patterns of different model parallelism strategies, develop a profile-based analytical model for estimating training throughput and cost, and provide guidelines for allocating geo-distributed resources efficiently. We find that although ML training throughput and cost with pure data parallelism degrades significantly when nodes span geographic regions, cross-region training with pipeline parallelism is practical.

CCS Concepts: • **Computing methodologies** → **Machine learning**.

Keywords: Machine Learning, Cloud computing

ACM Reference Format:
Foteini Strati, Paul Elvinger, Tolga Kerimoglu, and Ana Klimovic. 2024. ML Training with Cloud GPU Shortages: Is Cross-Region the Answer?. In *4th Workshop on Machine Learning and Systems (EuroMLSys '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3642970.3655843

1 Introduction

Training or fine-tuning machine learning (ML) models with millions or billions of parameters requires multiple high-end hardware accelerators, such as GPUs. Since not all users have access to on-premise datacenters, many host their ML training jobs in the public cloud, where they can choose from a range of accelerators across a variety of availability zones and geographical regions. Typically, users deploy distributed ML jobs within a single availability zone to take advantage of fast interconnects between virtual machines (VMs).

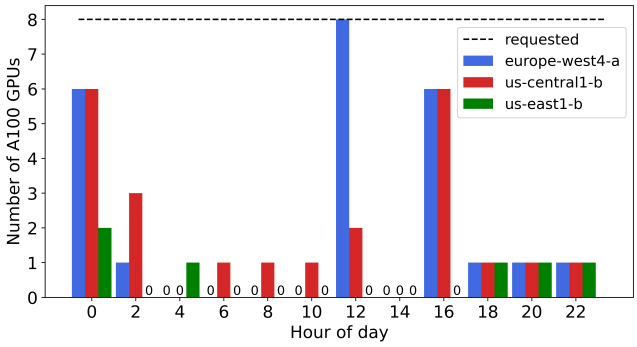


Figure 1. We request 8 A100 GPUs in 3 different zones in Google Cloud, re-attempting every 2 hours. Only a small fraction of GPU allocation requests are granted.

However, the widespread adoption of ML has led to a GPU shortage in the public cloud, making it difficult for users to get a hold of GPUs on-demand in a particular zone [41]. For example, Figure 1 shows how many A100-40GB GPUs we obtain in Google Cloud (GCP) over a 24 hour window when requesting 8 GPUs. We repeated the experiment every 2 hours and in three regions, always releasing the GPUs between experiments. Although we request 8 `a2-highgpu-1g` machines in each region, we acquire much fewer. This GPU scarcity has been acknowledged by other recent works [41].

A natural way to get more GPUs at a time in the cloud is to allocate GPUs across multiple zones and regions, depending on current availability. However, there are several challenges with training ML models across zones. First, when crossing availability zone boundaries, inter-VM network bandwidth decreases (e.g., we observe up to 8.05× in GCP) and network latency increases. Since many distributed ML training jobs are communication-bound [33], this can significantly impact training throughput. Second, cloud providers also charge for the volume of data sent across availability zone boundaries. For instance, in GCP, the cost per GB can vary from 0.01$ to 0.14$ [2]. Due to the massive size of ML models and the data they need to exchange across numerous iterations to reach convergence, the total amount of exchanged data can

EuroMLSys '24, April 22, 2024, Athens, Greece

ACM ISBN 979-8-4007-0541-0/24/04
https://doi.org/10.1145/3642970.3655843

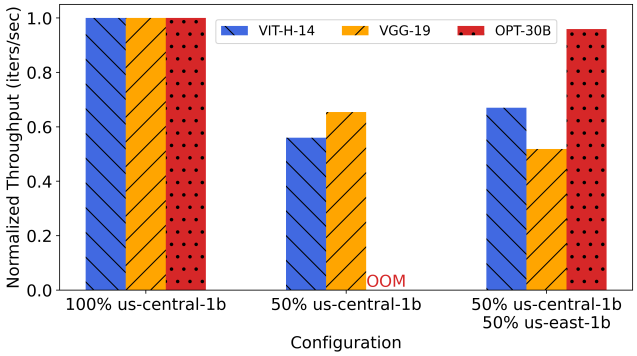


Figure 2. Examples of spreading distributed training across regions. VGG-19 and VIT-H-14 are trained with data parallelism, while OPT-30B with pipeline parallelism. Leveraging GPUs from different regions is beneficial for VGG-19 and OPT-30B, but not for VIT-H-14.

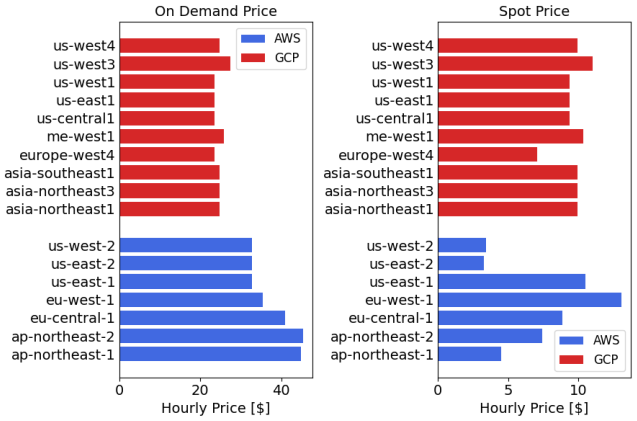


Figure 3. Average hourly spot and on-demand prices for 8-A100 GPUs in GCP and AWS in February 2024

reach up to TB, which leads to high costs. Since the volume of data exchanged between particular nodes of an ML job depends on the model architecture and the type of parallelism, optimizing cross-region training deployments must take these job characteristics into account. Finally, deciding which zones to use is further complicated by the fact that GPU prices can vary significantly across zones. Figure 3 shows up to 3.9× difference in A100-GPU prices in AWS.

Figure 2 shows examples of spreading training across 2 regions for 3 different models and 2 different parallelism techniques (data and pipeline parallelism). We compare the training throughput achieved when spreading across regions, and the training throughput when using fewer GPUs (50%) in a single region. While spreading across regions is beneficial for VIT and OPT, it leads to worse training throughput for the VGG model. Therefore, the benefits of spreading distributed training across regions depend on the model characteristics and parallelism strategy.

In this work, we ask the question: when and how does it make sense to use GPUs across zones and regions for large-scale, distributed training? We develop an analytical model that can accurately estimate training throughput and cost under various geo-distributed scenarios and parallelization strategies. First, we identify the key factors that influence the performance and cost of cross-region training, such as the ratio of computation to communication and the amount of exchanged data. Second, considering the various forms of ML training parallelism, we build a lightweight profiler and analytical model, that estimates training throughput and cost under a variety of configurations. Unlike prior work that focuses only on small models trained with data parallelism [22], or analyzes only training throughput ignoring cloud costs [42], our analysis encompasses both data and pipeline parallelism and examines the training of models of various scales from Computer Vision (CV) and Natural Language Processing (NLP) domains.

We find that spreading distributed training across zones within the same cloud region has minimal effects on training throughput since the network bandwidth is similar to single-zone setups (see Table 3). Across-region or across-continent training is detrimental for data parallelism but has only modest throughput degradation for pipeline parallelism. When both data and pipeline parallelism are used, we find it is helpful to distribute the different pipeline stages across regions, while maintaining data parallel traffic within one region.

2 Background and Motivation

2.1 Distributed DNN training

The communication patterns and data exchange volume in distributed ML training depend on the parallelism method and influence the optimal cross-region configuration. Here, we describe the main types of ML training parallelism:

Data Parallelism: During data parallelism, the dataset is partitioned among the available workers, and the model is replicated across the workers. After performing one or more forward and backward passes in local minibatches, the workers synchronize their gradients using techniques such as all-reduce [33].

Pipeline Parallelism: With pipeline parallelism, model layers are distributed to multiple GPUs. Each GPU (*stage*) holds one or more layers, performs forward and backward passes for its local layers, and sends and receives activations and gradients to and from its adjacent stages.

Tensor Model Parallelism: A model layer or operator, such as GEMM, is split across multiple GPUs. After each GPU performs its local computations, the results are reduced and propagated to the next layer. Tensor-model parallelism requires very fast interconnects between GPUs, and is most commonly limited within GPUs of a single node. [34]

Traffic Between	Cost/GB ($)	Latency (ms)	Bandwdith (GB/sec)
Same AZ (US)	Free	<1	1.45
Diff. AZ, same region (US)	0.01	0.9	1.42
Diff. regions (US)	0.02	31	0.63
Diff. continents (US/EU)	0.05	102	0.18

Table 1. Cost in February 2024, network latency and bandwidth for `a2-highgpu-1g` machines. We used NCCL [12] to measure network bandwidth and iperf3 [8] to measure network latency. The cost per GB can reach up to 0.16$ [4]

These parallelism types can be used independently or together. Multiple works focus on finding the optimal combination and degree of each parallelism type, either using 2D parallelism, i.e. a combination of data and pipeline parallelism [15] or 3D parallelism [27, 36]. While we focus on 2D parallelism, our findings also apply to 3D parallelism.

2.2 GPUs in the public cloud

Cloud providers [3, 7, 10] offer VMs across different availability zones, regions, and continents, at (potentially) different prices. This price difference can reach up to 3.9×, depending on the availability zone and the chosen provider (see Figure 3). Systems like SkyPilot [41] search the cloud providers' landscape for the most cost-effective VM that meets user-specific requirements. However, VM availability is not guaranteed, as shown in Figure 1, especially for transient (*spot*) VMs offered at 60-90% lower prices than on-demand VMs [6] but experiencing high preemption rates [14, 38].

The difference in GPU prices across cloud regions, and the scarcity of GPUs, motivate the usage of GPUs across any cloud region. However, there are two main limitations. First, when crossing cloud availability zones, the network bandwidth decreases, as shown in Table 1. The network bandwidth across VMs in different regions can be up to 8.05× lower than within the same zone. Second, cloud providers charge for data exchange between VMs in different availability zones and regions, with the cost per GB varying from 0.01$ to 0.05$ in our examined scenarios. Since distributed ML training involves across-worker communication (§2.1), both the drop in network bandwidth and the data exchange charges can negatively affect ML training jobs spanning more than one availability zone or region.

3 Modeling Cross-Region ML Training Performance and Cost

To analyze the potential benefits and challenges associated with cross-region ML training, we explore two key questions. First, when does it make sense to use cloud resources *across zones or regions* to train ML models? Second, how should ML model training jobs be *partitioned and parallelized* to minimize cross-region communication overheads?

Parameter	Description
mbs, gbs	Microbatch size, Global batch size
t_f,t_b,t_u	Time for forward, backward pass and update
ga	Number of Gradient accumulation steps
M	Model size
Act_i	Activation size of Layer i
$Grad_i$	Gradient size of Layer i

Table 2. Profiling information required for simulations

Parameter	Description
num_vm_i	Number of VMs in zone i
$cost_i$	Cost of VM (of a specific type) in zone i
$c_{i,j}$	Cost of exchanging 1 GB between 2 VMs in zones i and j

Table 3. Cloud-specific parameters

To answer these questions, we model ML training throughput and cost by taking into account the computation and communication characteristics of ML workloads and parallelization strategies and the characteristics of cloud deployments. Our methodology consists of two phases: profiling (§3.1) and simulations (§3.2). During profiling, we collect measurements for the different training phases and cloud characteristics. In the simulation phase, we model the time required for a training iteration. Based on this information, we estimate training throughput and cost.

3.1 Profiling

For a given model and GPU type, we collect measurements for its forward, backward, and update phase, by running a few iterations in the specific GPU type. For LLMs with repetitive layer structures (e.g. the transformer layer [16]), it is enough to profile this layer only once. Thus, we can profile large models (such as OPT-30) on 1 GPU. Apart from timing measurements, we also get the number of parameters for the model, the activation size of each layer, the microbatch size per GPU, and the global batch size used. Table 2 summarizes the required information for each model. Finally, we measure the inter-VM network bandwidth for each use case and collect the cost per VM, and data exchange (see Table 3). To measure the inter-VM network bandwidth we used the *sendrecv* benchmark from the NCCL test suite [11] with `a2-highgpu-1g` VMs. As the bandwidth is dependent on the amount of data being transferred, we repeated the measurements for varying data sizes ranging from 64KB to 2GB. Next, we fit a sigmoid function on our measurements that our simulator uses to extrapolate the bandwidth for a given data size.

3.2 Throughput and cost simulations

Running in large-scale clusters to evaluate each possible scenario would be prohibitively expensive. Therefore, we build a simulator to model the training throughput (iterations/sec)

and cost (USD/iteration) based on our profiling information. We now describe the basic principles of our simulator design.

3.2.1 Throughput estimation. Training throughput depends on per-GPU computation and communication patterns across GPUs, which depend on parallelism type. For instance, as described in section 2.1, data parallelism includes an all-reduce step among all workers at the end of each iteration. On the other hand, with pipeline parallelism, gradients and activations are exchanged among *adjacent* pipeline stages.

Data Parallelism: We start by computing the time per iteration, by partitioning it into 2 phases, computation (t_{comp}), and communication (t_{comm}). The computation time is:

$$t_{comp} = (t_f + t_b) * ga + t_u$$

where the number of gradient accumulation steps is calculated by $ga = \frac{gbs}{N \cdot mbs}$ with N being the number of workers. We follow ring-based All-Reduce for estimating the communication time [25, 32]. Assuming a model of M GB, each worker i will transmit $\frac{2 \cdot (N-1) \cdot M}{N}$ GB to worker $(i+1)\%N$. Since workers can exchange data in parallel to each other, the time for All-Reduce will be bottlenecked by the slowest network bandwidth b_{min} between any two workers.

Thus, the communication time will be:

$$t_{comm} = 2 \cdot (N-1) \cdot \frac{M}{N \cdot b_{min}}$$

Frameworks such as PyTorch [31] overlap gradient exchange with gradient computation. In the case of multiple gradient accumulation steps, the gradient exchange is overlapped with the backward pass at the final step. In the ideal case, where all GPUs are available in the same zone, the time for one training iteration with overlapping gradient exchange will be:

$$\begin{aligned} T_i = &(t_f + t_b) \cdot (ga - 1) \\ &+ max((t_f + t_b), 2 \cdot (N-1) \cdot \frac{M}{N \cdot b}) + t_u \end{aligned} \tag{1}$$

Assuming $\lambda \cdot N$ GPUs are available ($\lambda < 1$) we have:

$$\begin{aligned} T_\lambda = &(t_f + t_b) \cdot (\frac{ga}{\lambda} - 1) \\ &+ max((t_f + t_b), 2 \cdot (\lambda \cdot N - 1) \cdot \frac{M}{N \cdot \lambda \cdot b}) + t_u \end{aligned} \tag{2}$$

If we leverage GPUs from different regions, and the minimum network bandwidth is b_{min}, one training iteration is:

$$\begin{aligned} T_j = &(t_f + t_b) \cdot (ga - 1) \\ &+ max((t_f + t_b), 2 \cdot (N-1) \cdot \frac{M}{N \cdot b_{min}}) + t_u \end{aligned} \tag{3}$$

We are interested in the case where $T_j < T_\lambda$. Formulas (2) and (3) have a *max* term of 2 factors, resulting in 4 possible scenarios for $T_j < T_\lambda$. We consider the following 2 scenarios (as the rest can be straightforwardly handled):

Scenario 1:

$$\begin{aligned} T_j < T_\lambda \Rightarrow &(t_f + t_b) \cdot (ga - 1) + 2 \cdot (N-1) \cdot \frac{M}{N \cdot b_{min}} + t_u \\ &< (t_f + t_b) \cdot (\frac{ga}{\lambda} - 1) + 2 \cdot (\lambda \cdot N - 1) \cdot \frac{M}{N \cdot \lambda \cdot b} + t_u \end{aligned}$$

Scenario 2:

$$\begin{aligned} T_j < T_\lambda \Rightarrow &(t_f + t_b) \cdot (ga - 1) + 2 \cdot (N-1) \cdot \frac{M}{N \cdot b_{min}} + t_u \\ &< (t_f + t_b) \cdot (\frac{ga}{\lambda} - 1) + t_f + t_b + t_u \end{aligned}$$

We omit the intermediate calculations for brevity. Scenario (1) and (2) result in formulas 4 and 5 respectively.

$$b_{min} = \frac{1}{\frac{t_{fb}}{M} \cdot H + G} \tag{4}$$

$$b_{min} = \frac{M}{t_{fb}} \cdot C \tag{5}$$

where $t_{fb} = t_f + t_b$, and C, H, G are constants depending on λ, N, b, ga.

Equations 4 and 5 provide a lower bound for the network bandwidth, which serves as a criterion for distributing training across zones or regions. The key factors that determine whether training with more GPUs but with reduced bandwidth (across zones/regions) is more beneficial than training with fewer GPUs (within one zone) are the amount of per-GPU compute time per iteration (t_{fb}) and the size of the model (M). For instance, training can accommodate lower bandwidths when compute time per iteration is large. On the other hand, a larger model size exacerbates the impact of model synchronization, causing training throughput to significantly degrade with lower bandwidth.

Pipeline Parallelism: We use AMP's [27] formula to model the training of large models with pipeline parallelism. We define P to be the number of pipeline stages. During the forward pass, stage j sends the activation of its last layer to its successor stage $j+1$, while during the backward pass it receives the gradient of $j+1$'s first layer. Note that the last stage does not forward any activations and similarly the first stage does not forward any gradients. Let t_f^i and t_b^i be the time it takes to calculate the activations and gradients at stage i. We define $t_{comp}^i = t_f^i + t_b^i$ to be the computation time at stage i. Next we define the time spent on communication t_{comm}^i between stage i and $i+1$ to be the time it takes to send the activations and gradients between the two stages. When $b_{i,i+1}$ is the bandwidth between stage i and $i+1$ for $i \in [0, P-2]$, we get that $t_{comm}^i = \frac{Act_i + Grad_{i+1}}{b_{i,i+1}}$. As a last step, let us substitute the straggler term in AMP's formula as follows (nm is the number of microbatches processed at each iteration)

$$straggler = (nm - 1) \cdot \max_{\substack{0 \le i \le P-1 \\ 0 \le j \le P-2}} (t_{comp}^i, t_{comm}^j)$$

Model	Number of parameters	Dataset	Global batch size	Optimizer	Source
OPT-1.3B,6.7B,30B	1.3B,6.7B,30B	WikiText	1M,2M,4M tokens	Adam	[43]
VGG-19-BN	144M	ImageNet	8192	SGD	[14, 35]
Vit-H-14	632 M	ImageNet	4096	Adam	[17]
ConvNext-Large	197M	ImageNet	4096	Adam	[29]

Table 4. Examined models. We selected the global batch size and optimizer used in the respective paper.

Using the above defined quantities and the collected information from tables 2 and 3, we extend AMP's formula as follows:

$$\begin{aligned} t_{pp} &= straggler + \sum_{i=0}^{P-1} t^i_{comp} + \sum_{i=0}^{P-2} t^i_{comm} + t_u \\ &= straggler + \sum_{i=0}^{P-1} (t^i_f + t^i_b) + \sum_{i=0}^{P-2} \frac{Act_i + Grad_{i+1}}{b_{i,i+1}} + t_u \end{aligned} \tag{6}$$

The formula assumes that communication can be overlapped with computation, as in multiple works [5, 20, 27].

Data + Pipeline Parallelism: When both data and pipeline parallelism are used, the training throughput depends on the number of pipeline replicas D, and the number of stages per pipeline P [15, 27, 36]. Assume we have D pipelines, each with depth P. We use equation 6 to estimate the iteration time of each pipeline. At the end of each iteration, the data parallel replicas need to synchronize their updates (using All-Reduce). Assuming balanced partitioning of the model at each stage, the time for replica synchronization will be: $t_{sync} = 2 \cdot (D-1) \cdot \frac{M}{D \cdot P \cdot b_{min}}$. Thus, the iteration time will be $t_{iter} = t_{pp} + t_{sync}$.

We analyze two possible ways of splitting a $D \times P$ grid: (1) on the data parallel dimension D and (2) on the pipeline parallel dimension P. In the first case, sending activations and gradients between pipeline stages happens within the same availability zone. In the second case, the AllReduce operation to synchronize model weights is done across workers of the same availability zone.

3.2.2 Cost estimation.

To get the cost per iteration, we again divide it into two parts: computation cost (C_{comp}), and communication cost (C_{comm}). The computation cost will be (regardless of the type of parallelism):

$$C_{comp} = t_{iter} * \sum_{\text{zone } i} (num\_vm_i \cdot cost_i)$$

The communication cost is $C_{comm} = \sum data_{ij} \cdot c_{ij}$, where $data_{ij}$ is the amount of data exchanged between VMs i and j. Note that both $data_{ij}$ and c_{ij} can be zero for a pair of VMs.

Data Parallelism: With a ring-based All-reduce, each worker i will send data to the $(i+1)\%N$-th worker. The amount of data exchanged for the whole All-Reduce operation will be:

$$\underset{0 \le i \le N-1}{data_{ij}} = \begin{cases} \frac{2 \cdot (N-1) \cdot M}{N}, & \text{if } j = (i+1)\%N \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Pipeline Parallelism: During forward pass, each worker i will send activations of the last layer to worker $i+1$, and during backward pass, will receive gradients from worker i+1 for $i \in [0, P-2]$. Thus, for one training iteration, assuming that nm microbatches are being processed, we have:

$$\underset{0 \le i \le P-2}{data_{ij}} = \begin{cases} nm \cdot (Act_i + Grad_j), & \text{if } j = i+1 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Data + Pipeline Parallelism: Assuming each pipeline replica spans an entire row, we assign each worker a rank in row-major order. Worker i will exchange data with its adjacent workers both during forward and backward passes and during gradient synchronization:

$$\underset{0 \le i \le DP-1}{data_{ij}} = \begin{cases} nm \cdot (Act_i + Grad_j), & \text{if } j = i+1, \text{ and} \\ & i \notin \{kP-1 \mid 1 \le k \le D\} \\ \frac{2 \cdot (D-1) \cdot \frac{M}{P}}{D}, & \text{if } j = (i+P)\%(D \cdot P) \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

4 Evaluation

In this section, we evaluate the impact of across-zone or region training on various ML models.

4.1 Setup

ML workloads. Table 4 shows the models we profiled, which range from 144M to 30B parameters, from computer vision to natural language processing domains. We get the global batch sizes from the papers that propose these models. We used PyTorch 2.1 and DeepSpeed (where pipeline parallelism is used) for profiling. When using a combination of data and pipeline parallelism, we select the partitioning strategy that leads to the highest training throughput [27].

Cloud setup. We profiled our models and got network bandwidth information using `a2-highgpu-1g` VMs with 1 40GB-A100 GPU attached. We run experiments across 4 different availability zones (*us-central1-b*, *us-west1-b*, *us-east1-b*, and *europe-west4-a*) in GCP. We fix the number of GPUs in each experiment and compare performance and cost when the GPUs are spread across 1, 2, and 3 zones.

We validated our methodology by comparing the estimated throughput with the actual throughput of various parallelization strategies across different zones and regions. Our estimations are within 10% of the actual throughput.

4.2 Training with Data Parallelism only

Figure 4 plots estimated training throughput and cost for the VGG, ConvNext, and VIT models on 32 A100 GPUs. Spreading training in two zones in the same region always leads to higher throughput compared to 75% or 50% of the requested machines in one availability zone. The network bandwidth across zones is within 95% of the network bandwidth within one zone, so training can still maintain high throughput.

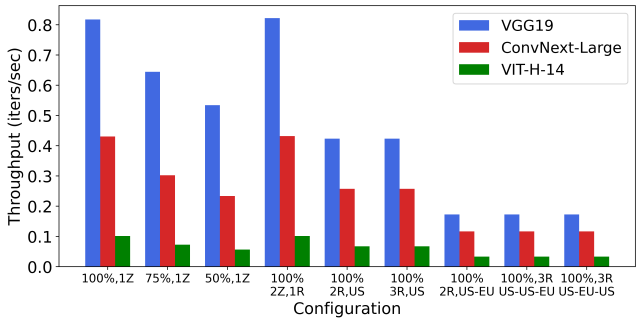
When using two or more regions in the US, the impact on training throughput depends on model characteristics. The bandwidth is within 44% of the intra-zone network bandwidth. Distributing training across regions leads to higher throughput for ConvNext and VIT, but not for VGG19, compared to using 50% of GPUs in a single zone. According to Formulas 4 and 5, VIT and ConvNext exhibit higher $\frac{t_{fb}}{M}$ compared to VGG (i.e. compute-to-communication ratio), resulting in a lower minimum bound for network bandwidth and thus benefiting from spreading across regions. Intercontinental training (i.e. US-EU) has detrimental effects on the training throughput of all models. The network bandwidth is within 14% of the intra-zone bandwidth, far lower than the acceptable lowest bound of any of the models. When crossing the region or continent boundaries, all-reduce is bottlenecked by the lowest network bandwidth link, thus there is no difference in throughput when using more than two regions.

Figure 4b shows the cost for 100 iterations, divided into VM rental cost, and communication cost. The VM cost increases as we go from a single zone to multiple zones, regions, and continents, due to the gradual decrease in training throughput. As expected also from Table 1, the cost for data exchange is very high for inter-continental setups, due to the high charges per GB. When using more regions, the cost for data exchange also increases, since more links are charged, as shown from Figure 4b, where the 3-region setups incur higher costs than the respective 2-region cases.

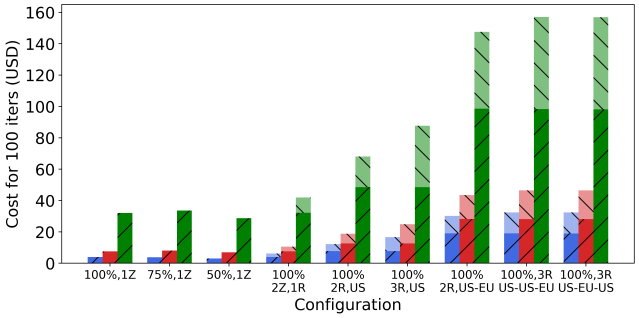
4.3 Training with Pipeline Parallelism only

Figure 5 includes our estimations for OPT-30B on a cluster of 25 A100 GPUs. We select the minimum number of machines to fit the model, thus we exclude cases where we use only 50% or 75% of machines in a single zone, that would lead to out-of-memory errors. Training throughput remains almost unaffected, regardless of whether we split across zones within the same region, across regions within the same continent, or even across continents.

As explained in section 3, when employing pipeline parallelism, workers at adjacent stages exchange activations



(a) Training Throughput

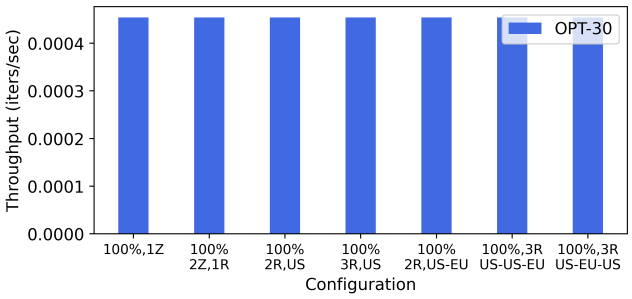


(b) Cost for 100 iterations. Bold bars stand for VM rental cost, faint bars stand for communication cost.

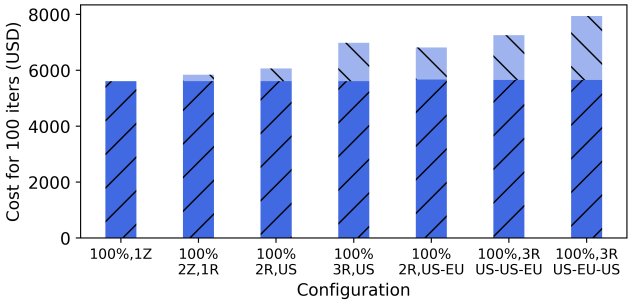
Figure 4. Simulations with data parallelism. We use local batch size (per GPU) 128 for VGG19, 64 for ConvNext-Large, and 32 for VIT-H-14.

(and gradients) of the last (or first) layer of each stage. Even though the examined models are at the scale of GB, the sizes of per-layer activations are within 10s of MB per sample. Consequently, the time to exchange the activations and gradients is small, even under a tighter across-region network bandwidth and can be efficiently overlapped by the compute time required for forward and backward passes per microbatch.

To analyze cost, we assume that the P pipeline stages are spread across N zones. Inter-stage communication thus traverses zone boundaries at $N-1$ points in the pipeline. This is a valid assumption, as users can control the rank assignment of workers. The VM rental cost remains very similar across all configurations, differing by at most 1%. This is due to the nearly identical rental costs in the respective regions and the fact that the distribution of the pipeline across different locations has negligible impact on training throughput, and consequently training time. The main cause of diverging training costs across different configurations is data movement cost. Spreading training across zones in the same region has minimal effects (up to 4%) on training costs. Due to higher charges per GB movement, the training cost across regions and continents is 25% and 42% higher

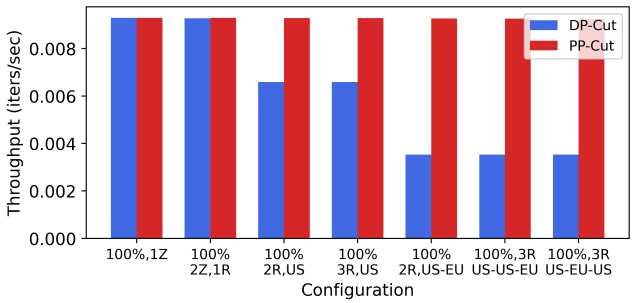


(a) Training Throughput

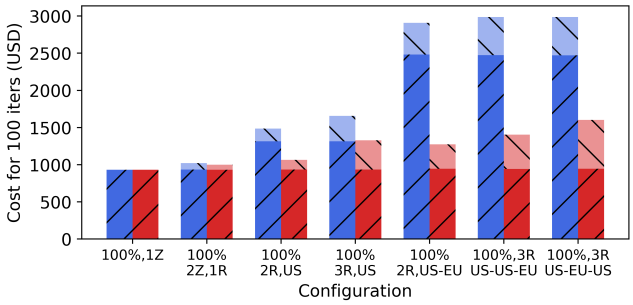


(b) Cost for 100 iterations. Bold bars show VM rental cost, faint bars show data transfer costs.

Figure 5. Simulations running OPT-30B on a cluster of 25 A100 GPUs with Pipeline Parallelism. The simulation sets $P = 25$ stages, assigns 2 layers per stage and uses a micro-batch size of 1.



(a) Training Throughput



(b) Cost for 100 iterations. Bold bars stand for VM rental cost, faint bars stand for communication cost.

Figure 6. Simulations for running OPT-6.7B on a cluster of 85 A100 GPUs with 2D Parallelism. We use $D = 5$, $P = 17$, assign 2 layers per stage and use a microbatch size of 1.

than single-zone respectively. Notably, the order of stage assignment affects costs. Comparing the *US-US-EU* and *US-EU-US* cases in Figure 5b, we observe that the *US-EU-US* setup leads to higher costs. Since each stage exchanges data with its adjacent stage, and across-region data exchange costs are lower than across-continents costs (see Table 1), it is important to minimize the number of crossing continent boundaries.

4.4 Training with 2D Parallelism

Figure 6 presents our training estimations for OPT-6.7B on a cluster of 85 A100 GPUs. We aim to determine which partitioning strategy (data or pipeline parallel) is more beneficial. When distributing training in two zones of the same region, both strategies show similar throughput to the single-zone throughput. When using GPUs from different regions and continents, splitting training on the pipeline parallel dimension is more beneficial. This aligns with our observations in sections 4.2 and 4.3. In section 4.2 we noted a significant performance drop in data-parallel throughput, especially in the EU-US setup, due to all-reduce overheads. On the other hand, the effects of lower network bandwidth can be better hidden with pipeline parallelism, as shown section 4.3.

We make similar observations for training cost: partitioning along the data parallel dimension results in higher costs, both due to lower training throughput and data exchange charges. In summary, when training with 2D parallelism, keeping all data parallel communication within one region leads to higher throughput and lower cost.

5 Discussion

ML model and hardware trends: A question that might arise from our study, is how temporary this increased GPU demand, and resulting GPU scarcity in the public cloud is. The high GPU demand is a result of the great advances in the ML field over the recent years. Figure 7 plots the LLM size (using fp16) and the maximum memory capacity of NVIDIA GPUs released from 2018 to 2023. Over 5 years, we observe that, while the model size has experienced a 4400× increase, the increase in the available per-GPU memory is only 6×. Consequently, the number of GPUs required for training and serving these models is drastically increasing over time, resulting in GPU scarcity that we and others have observed [41]. Given these trends, we believe that the demand for accelerators will continue to increase, intensifying the GPU scarcity in the public cloud. Opportunistically allocating resources across multiple cloud zones will facilitate

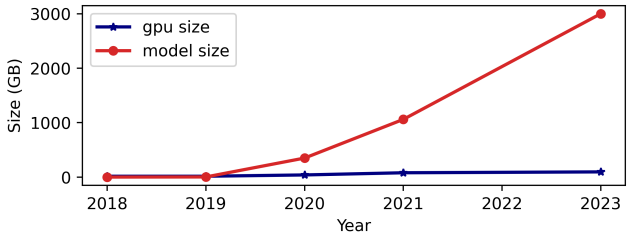


Figure 7. The evolution of LLM size and GPU memory from 2018 to 2023. Data from [1, 9, 40].

the training of such large models. We hope our analysis will help guide optimal resource allocation.

Availability of spot instances: In this paper we focused on on-demand VMs. Another type of VMs available in the public cloud is *spot* or *preemptble* VMs, offered at 60-90% of the price of on-demand VMs [6]. However, spot instances can be preempted at any time depending on resource demand and availability. Recent studies indicate that GPU-equipped VMs experience high rates of preemptions [14, 26, 38, 41]. This scarcity and high preemption rates may lead users to spread distributed ML training across multiple zones. Our analysis aims to provide insights on when and how to allocate spot instances across cloud zones (since the computation and communication patterns remain the same as with on-demand machines). An additional factor that needs to be considered is the preemption rate, which can severely affect the training iteration time, as reconfiguration is required upon a preemption event [15, 18, 23, 38], which can take a significant amount of time [15, 18, 23]. Preemption rates tend to vary across cloud zones [38]. We plan to include preemption rates in our analysis, to help users make more informed decisions based on the expected system behavior.

6 Related Work

Geodistributed ML training: Frameworks facilitating geo-distributed ML training are emerging [37, 42]. Yuan et al [42] study geo-distributed training of foundation models across low-bandwidth networks and heterogeneous devices, analyzing various placement policies. However, they do not consider the monetary cost of exchanging data, which is a fundamental decision factor, as highlighted in our work. Some of these frameworks propose asynchronous training [21, 28, 44], or quantization and compression to reduce the amount of exchanged data [13, 19, 24, 39] in geo-distributed setups. Since synchronous training offers stronger convergence guarantees, it is most widely used [15, 30], and is the main focus of our work.

Towards automating resource allocation in the cloud A couple of works automate resource allocation in the cloud, considering prices and resource availability. DeepSpot [26] monitors GPU prices and recommends optimal regions for training. SkyPilot [41] is a cloud broker that distributes tasks across cloud providers and regions. However, both tools constrain training to a single availability zone. As highlighted in our work, leveraging GPUs from various zones and regions is essential for efficient distributed training at scale.

Study of across-region training in the cloud Erben et al [22] investigate geo-distributed ML training in the public cloud. They focus on small models, trained with data parallelism. They introduce the *granularity* metric, defined as the ratio of computation to communication, and show that models with higher granularity are more suitable for geo-distributed training. However, they do not provide a methodology on how to calculate granularity for a model. In contrast, we provide analytical models, that, based on basic profiling information can accurately estimate training throughput and cost. Additionally, our analysis includes large models, trained with pipeline parallelism, which exhibits different communication patterns compared to data parallelism.

7 Conclusion

We studied distributed ML training across various cloud regions, to address GPU shortages. We proposed an analytical model, that, based on model and cloud characteristics, can accurately estimate training throughput and cost under various setups. We analyzed ML models of various scales and different parallelization strategies. We find that pipeline parallel training is more tolerant of geo-distributed training than data-parallel training. This determines how to optimally allocate resources in large-model training setups using a combination of data and pipeline parallelism. We plan to implement a scheduler to optimize resource allocation across cloud regions based on our findings.

Acknowledgement

We thank our anonymous reviewers for their valuable feedback. We thank Ixeia Sánchez Périz for her help throughout the project. Foteini Strati is supported by the Swiss National Science Foundation (Project Number 200021_204620).

References

[1] 2023. Train and deploy large language models on Amazon SageMaker. https://d1.awsstatic.com/events/Summits/reinvent2022/AIM405_Train-and-deploy-large-language-models-on-Amazon-SageMaker.pdf.
[2] 2024. All networking pricing. https://cloud.google.com/vpc/network-pricing.
[3] 2024. Amazon Web Service. https://aws.amazon.com/.
[4] 2024. Azure Bandwidth Pricing. https://azure.microsoft.com/en-us/pricing/details/bandwidth/#pricing.
[5] 2024. DeepSpeed Pipeline Parallelism. https://www.deepspeed.ai/tutorials/pipeline/.
[6] 2024. GCP Spot VMs. https://cloud.google.com/spot-vms.
[7] 2024. Google Cloud Platform. https://cloud.google.com/?hl=el.
[8] 2024. iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool. https://github.com/esnet/iperf.

[9] 2024. List of Nvidia graphics processing units. https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units.

[10] 2024. Microsoft Azure. https://azure.microsoft.com/en-us.

[11] 2024. NCCL Tests. https://github.com/NVIDIA/nccl-tests.

[12] 2024. NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl.

[13] Syeda Nahida Akter and Muhammad Abdullah Adnan. 2020. WeightGrad: Geo-Distributed Data Analysis Using Quantization for Faster Convergence and Better Accuracy. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 546–556. https://doi.org/10.1145/3394486.3403097

[14] Joel André, Foteini Strati, and Ana Klimovic. 2022. Exploring learning rate scaling rules for distributed ML training on transient resources. In *Proceedings of the 3rd International Workshop on Distributed Machine Learning* (Rome, Italy) *(DistributedML '22)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3565010.3569067

[15] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. 2022. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems* (Rennes, France) *(EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 472–487. https://doi.org/10.1145/3492321.3519584

[16] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*. https://openreview.net/forum?id=YicbFdNTTy

[18] Jiangfei Duan, Ziang Song, Xupeng Miao, Xiaoli Xi, Dahua Lin, Harry Xu, Minjia Zhang, and Zhihao Jia. 2024. Parcae: Proactive, Liveput-Optimized DNN Training on Preemptible Instances. arXiv:2403.14097 [cs.DC]

[19] Chenyu Fan, Xiaoning Zhang, Yangming Zhao, Yutao Liu, and Shui Yu. 2023. Self-Adaptive Gradient Quantization for Geo-Distributed Machine Learning Over Heterogeneous and Dynamic Networks. *IEEE Transactions on Cloud Computing* 11, 4 (2023), 3483–3496. https://doi.org/10.1109/TCC.2023.3292525

[20] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. 2021. DAPPLE: a pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 431–445. https://doi.org/10.1145/3437801.3441593

[21] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 629–647. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh

[22] Alexander Isenko, Ruben Mayer, and Hans-arno Jacobsen. 2023. How Can We Train Deep Learning Models Across Clouds and Continents? An Experimental Study.

[23] Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury. 2023. Oobleck: Resilient Distributed Training of Large Models Using Pipeline Templates. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. ACM. https://doi.org/10.1145/3600006.3613152

[24] Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. 2018. SketchML: Accelerating Distributed Machine Learning with Data Sketches. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1269–1284. https://doi.org/10.1145/3183713.3196894

[25] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 463–479. https://www.usenix.org/conference/osdi20/presentation/jiang

[26] 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

[27] Dacheng Li, Hongyi Wang, Eric Xing, and Hao Zhang. 2022. AMP: Automatically Finding Model Parallel Strategies with Heterogeneity Awareness. arXiv:2210.07297 [cs.LG]

[28] Shigang Li, Tal Ben-Nun, Giorgi Nadiradze, Salvatore Di Girolamo, Nikoli Dryden, Dan Alistarh, and Torsten Hoefler. 2021. Breaking (Global) Barriers in Parallel Stochastic Optimization With Wait-Avoiding Group Averaging. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2021), 1725–1739. https://doi.org/10.1109/TPDS.2020.3040606

[29] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A ConvNet for the 2020s. arXiv:2201.03545 [cs.CV]

[30] Andrew Or, Haoyu Zhang, and Michael None Freedman. 2022. VirtualFlow: Decoupling Deep Learning Models from the Underlying Hardware. In *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*, Diana Marculescu, Yuejie Chi, and Carole-Jean Wu (Eds.). mlsys.org. https://proceedings.mlsys.org/paper/2022/hash/2723d092b63885e0d7c260cc007e8b9d-Abstract.html

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach 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. arXiv:1912.01703 [cs.LG]

[32] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel and Distrib. Comput.* 69, 2 (2009), 117–124. https://doi.org/10.1016/j.jpdc.2008.09.002

[33] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtarik. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 785–808. https://www.usenix.org/conference/nsdi21/presentation/sapio

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

[35] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[36] Jakub M Tarnawski, Deepak Narayanan, and Amar Phanishayee. 2021. Piper: Multidimensional Planner for DNN Parallelization. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 24829–24840. https://proceedings.neurips.cc/paper_files/paper/2021/file/d01eeca8b24321cd2fe89dd85b9beb51-Paper.pdf

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

[38] John Thorpe, Pengzhan Zhao, Jonathan Eyolfson, Yifan Qiao, Zhihao Jia, Minjia Zhang, Ravi Netravali, and Guoqing Harry Xu. 2023. Bamboo: Making Preemptible Instances Resilient for Affordable Training of Large DNNs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 497–513. https://www.usenix.org/conference/nsdi23/presentation/thorpe

[39] Jue Wang, Yucheng Lu, Binhang Yuan, Beidi Chen, Percy Liang, Christopher De Sa, Christopher Re, and Ce Zhang. 2023. CocktailSGD: fine-tuning foundation models over 500mbps networks. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 1497, 19 pages.

[40] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. 2023. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. arXiv:2304.13712 [cs.CL]

[41] 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

[42] Binhang Yuan, Yongjun He, Jared Quincy Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy Liang, Christopher Re, and Ce Zhang. 2023. Decentralized Training of Foundation Models in Heterogeneous Environments. arXiv:2206.01288 [cs.DC]

[43] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL]

[44] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. 2016. Staleness-aware Async-SGD for Distributed Deep Learning. arXiv:1511.05950 [cs.LG]