Distributed Deep Learning In Open Collaborations

Michael Diskin[*♡†] **Alexey Bukhtiyarov**[*†♣] **Max Ryabinin**[*†♡]

Lucile Saulnier[‡] **Quentin Lhoest**[‡] **Anton Sinitsin**[†♡] **Dmitry Popov**[†♡]

Dmitry Pyrkin[♡] **Maxim Kashirin**[♡] **Alexander Borzunov**[†♡]

Albert Villanova del Moral[‡] **Denis Mazur**[♣] **Ilia Kobelev**[†♣] **Yacine Jernite**[‡]

Thomas Wolf[‡] **Gennady Pekhimenko**[♢♠]

† Yandex, Russia
‡ Hugging Face, USA
♡ HSE University, Russia
♣ Moscow Institute of Physics and Technology, Russia
♢ University of Toronto, Canada
♠ Vector Institute, Canada

Abstract

Modern deep learning applications require increasingly more compute to train state-of-the-art models. To address this demand, large corporations and institutions use dedicated High-Performance Computing clusters, whose construction and maintenance are both environmentally costly and well beyond the budget of most organizations. As a result, some research directions become the exclusive domain of a few large industrial and even fewer academic actors. To alleviate this disparity, smaller groups may pool their computational resources and run collaborative experiments that benefit all participants. This paradigm, known as grid- or volunteer computing, has seen successful applications in numerous scientific areas. However, using this approach for machine learning is difficult due to high latency, asymmetric bandwidth, and several challenges unique to volunteer computing. In this work, we carefully analyze these constraints and propose a novel algorithmic framework designed specifically for collaborative training. We demonstrate the effectiveness of our approach for SwAV and ALBERT pretraining in realistic conditions and achieve performance comparable to traditional setups at a fraction of the cost. Finally, we provide a detailed report of successful collaborative language model pretraining with 40 participants.

1 Introduction

The deep learning community is becoming increasingly more reliant on transfer learning. In computer vision, pretraining convolutional networks on large image collections such as ImageNet [1] is the de facto standard for a wide range of applications ranging from object detection [2] and semantic segmentation [3] to image classification [4] and even learning perceptual similarity [5]. A growing number of natural language processing systems capitalize on language models with billions of parameters [6, 7, 8, 9, 10, 11] trained on vast unlabeled corpora. Similar trends have emerged in areas such as speech processing [12], reinforcement learning [13], and computational biology [14, 15].

[*]Equal contribution. Correspondence to `mryabinin0@gmail.com`
Detailed author contributions are listed at the end of the work.

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

Training these models is a notoriously time-consuming and challenging task: it often requires hundreds of high-end GPU servers [10, 16] and would take multiple years on a single device [17]. Most academic and independent researchers simply cannot afford to train state-of-the-art models from scratch, which slows down scientific progress and practical adoption of deep learning.

Historically, the deep learning community has addressed this problem via "model hubs" or "model zoos" — public repositories for pretrained model checkpoints [18, 19, 20, 21]. These repositories have played a significant role in the democratization of deep learning, allowing everyone to reap the benefits of large-scale training runs conducted by corporations and universities with sufficient resources. However, model hubs are limited to a narrow subset of datasets and tasks that match the interests of model creators. For instance, in natural language processing, it is often difficult to find up-to-date models for more than a handful of languages [22]. In turn, computer vision hubs rarely feature models trained on drawings, satellite images, 3D renders, microscopy, or any other data that does not resemble ImageNet. As a result, many researchers in these areas can only work on problems for which there are available pretrained models rather than the problems that most need solving.

However, there might be an alternative way to obtain pretrained models: to train these models *collaboratively*. This approach, known as volunteer (or grid) computing, allows many independent parties to combine their computational resources and collectively perform large-scale experiments [23, 24, 25]. The raw compute performance of such collaborations often exceeds that of the fastest supercomputers [26]; however, fully utilizing it can be challenging due to several reasons. First, devices that contribute to collaborative experiments can range from GPU servers and high-end workstations to consumer-grade computers and even smartphones [27]. Second, most of these devices use household internet connection with limited bandwidth and low reliability. Third, participants in such projects often donate their hardware part-time, joining and leaving the experiment at will.

While it is theoretically possible to train neural networks on this kind of infrastructure, modern distributed training strategies are only efficient in a narrow range of conditions. For instance, training with Ring All-Reduce [28] works well for identical servers but suffers significant performance penalties from network latency or bandwidth variation [29]. Another technique known as Parameter Server can handle heterogeneous devices at the cost of being less scalable [30]. Applying any of these strategies outside their preferred conditions may significantly reduce the training throughput [31], which makes them difficult to apply in the volatile infrastructure of volunteer computing. This issue is further complicated by the unique limitations of volunteer devices, such as network address translation (NAT), regional access restrictions, or variations in performance.

In this study, we carefully analyze the above challenges and come up with a practical solution for **D**istribut**e**d **D**eep **L**earning in **O**pen **C**ollaborations (DeDLOC). DeDLOC is based on a novel algorithm that adapts to the available hardware in order to maximize the training throughput. Depending on the infrastructure, DeDLOC can recover parameter servers [30], All-Reduce SGD [32], decentralized SGD [33], BytePS [34], or an intermediate strategy that combines all of them. Using this algorithm, we propose a system for collaborative training designed to accommodate a large number of heterogeneous devices with uneven compute, bandwidth, reliability, and network capabilities.

The contributions of our work can be summarized as follows:

- We analyze the unique challenges of distributed training in open collaborations and propose a practical recipe for training in these conditions.
- We formulate a novel distributed training algorithm that interpolates between traditional strategies to directly maximize the training performance for the available hardware.
- We verify the effectiveness of the proposed algorithm and system design for unsupervised pretraining of ALBERT-Large and SwAV under realistic conditions.
- We run collaborative training with actual volunteers, achieving competitive results to models trained on hundreds of data center GPUs. We also report insights on the collaborator activity and share the codebase for running similar experiments in the future[2].

[2]Code and training configurations are available at `github.com/yandex-research/DeDLOC`

2 Related work

2.1 Distributed training

In this work, we focus on distributed data-parallel training, where each device runs forward and backward pass of the entire model on a subset of training examples. While there are many alternative techniques [35, 36, 37], data-parallel is still the most popular strategy. Even the model-parallel approaches for extremely large models rely on data parallelism at the top level [37, 16, 38].

Training on multiple nodes was first implemented with parameter server (PS) [30]. This training strategy relies on a dedicated node that stores model parameters and executes optimization steps using the gradients sent by workers. In turn, worker nodes iteratively download the latest version of model parameters from the server, compute gradients and submit them back to the PS. This strategy is easy to implement and use, but it has an unavoidable bottleneck: the entire system performance is limited by the network throughput of a single server. Since then, the scientific community proposed numerous extensions to PS that alleviate the bottleneck by reducing the communication load [39, 40, 41, 42, 43], introducing asynchronous updates [44, 45] or training with multiple servers [46, 34].

The issue of uneven communication load has also inspired the development and widespread adoption of another group of methods that rely on All-Reduce for gradient averaging [47, 48, 49]. All-Reduce is a family of collective operations that allow nodes to efficiently aggregate (e.g. average) their local vectors and distribute the result across all devices [28, 50, 51]. Unlike parameter servers, All-Reduce assigns equal roles to all devices, making it easier to scale to a large number of homogeneous workers.

The popularity of AR-SGD sparked many practical applications for different scenarios. One particularly relevant application is elastic training [52, 53], which allows the user to add or remove workers at any point without interrupting the training run. While this bears a lot of similarity with collaborative training, we have found that elastic training systems are designed around global state synchronization, which makes them highly dependent on the homogeneity of the workers and their network connectivity. The overall efficiency is bounded by the performance of the lowest-performing node; as a result, introducing even a single low-bandwidth participant to such systems reduces the training speed by orders of magnitude.

Seeking to avoid the need for synchronization and centralized orchestration, the research community has developed decentralized training algorithms. These algorithms can be broadly divided into two categories: directly passing updates between peers [54, 55] or running All-Reduce in small alternating groups [56, 29]. Compared to PS and All-Reduce, both categories provide a greater degree of fault tolerance but often require more steps to converge due to delayed updates [33, 29].

Most practical use cases of the above techniques take place in HPC or cloud conditions, but there is one notable exception. In Federated Learning, multiple parties train a shared model on decentralized privacy-sensitive data that cannot be shared between devices [57]. For that reason, federated learning algorithms prioritize data privacy over training efficiency, often leaving most of the compute resources unused [58, 59]. For a more detailed overview of Federated Learning, refer to Appendix A.

2.2 Volunteer Computing

Volunteer computing (VC) is a paradigm of distributed computing where people donate the idle time of their desktops, smartphones, and other personal devices to solve a computationally hard problem collectively. This approach has seen successful applications in bioinformatics, physics and other scientific areas [60, 61, 62, 23, 63, 64, 65].

In all these applications, volunteer computing allows researchers to access vast computational resources. In Folding@home, over 700,000 volunteers have collectively contributed 2.43 exaFLOPs of compute to COVID-19 research in April of 2020 [26]. Another project named BOINC (Berkeley Open Infrastructure for Network Computing) brings together 41.548 petaFLOPs from over 790,000 active computers as of 17 March 2020 [25]. Volunteer computing systems were also the first "supercomputers" to reach 1 petaFLOP and 1 exaFLOP barriers [26, 66]. These results became possible due to the contributions of a broad range of devices from high-end workstations to smartphones and even gaming consoles [67].

Unfortunately, this compute diversity is also the main limitation of VC. Any volunteer computing system should be able to run on a wide range of available hardware and maintain integrity even if

some participants disconnect. Furthermore, the resources available to a project can vary over time, as most volunteers are only sharing their hardware when it is unused. Finally, volunteer devices are interconnected with a shared high latency network at typical home internet connection speeds.

As a result, there were only a few successful attempts to apply volunteer computing to machine learning workloads. One such project is MLC@Home [68], which relies on volunteers to train many small independent models. This specific problem can be solved with no direct communication between participants. By contrast, distributed training of a single model requires significantly more communication and does not allow a natural way to "restart" failed jobs. When it comes to distributed training of neural networks, most volunteer computing projects rely on parameter server architectures [69, 70, 71]. As a result, these systems are bounded by the throughput of parameter servers and the memory available on the weakest GPU. The only notable exception is Learning@home [72], which uses expert parallelism to train larger models spanning multiple computers; however, this approach has only been tested in simulated conditions.

3 Distributed Deep Learning in Open Collaborations

There are two unsolved challenges that stand in the way of practical collaborative training. The first challenge is algorithmic: how to maintain optimal training performance with dynamically changing hardware and network conditions? Another major challenge is ensuring consistent training outcomes with inconsistent composition of participants. Thus, we organize this section around these two issues:

- Section 3.1 provides a general overview of DeDLOC and explains how it maintains consistency in a dynamic environment.
- In Section 3.2, we describe the generalized communication strategy that maximizes training throughput by adapting to the currently available devices.
- In Section 3.3, we address system design challenges, such as circumventing NAT and firewalls, training on large datasets and managing collaborator access.

3.1 Ensuring training consistency

Many state-of-the-art models, notably GANs [73] and Transformers [74], require a strict training regimen. Deviating from the recommended batch size or introducing stale gradients may significantly affect the training outcome [75, 76, 77]. Since in a collaborative setting one has little control over the devices that participate in the experiment, it is almost guaranteed that the specific hardware setup will vary between runs and even during a single run. Without special precautions, these runs may result in models with vastly different final accuracy.

To avoid this pitfall, DeDLOC follows synchronous data-parallel training with fixed hyperparameters regardless of the number of collaborators. In order to compensate for relatively slow communication, we adopt training with extremely large batches [78, 49, 75, 79, 10], which allows peers to communicate less frequently. This strategy also provides a natural way to deal with heterogeneous hardware [80]: each device accumulates gradients at its own pace until the collaboration reaches the target batch size. Once ready, the collaborators exchange their gradients and perform one optimizer step. Using synchronous updates makes DeDLOC mathematically equivalent to large-batch training on a regular HPC cluster; see Appendix G for a more detailed explanation. Figure 1 gives a high-level visual explanation of this algorithm.

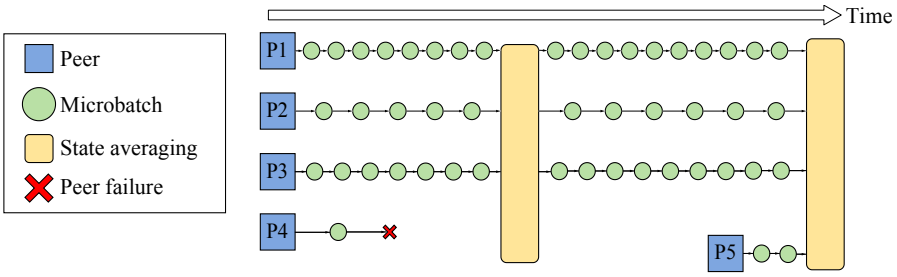


Figure 1: Two DeDLOC training iterations with example collaborator dynamics.

3.2 Adaptive averaging algorithm

As we discussed in Section 2.1, each distributed training algorithm has a narrow range of conditions where it can reach optimal performance. For instance, Ring All-Reduce works best on homogeneous hardware with low-latency communication, while Parameter Server strategy requires dedicated high-bandwidth devices that communicate with a large number of "workers". Since all devices are provided by volunteers, our training infrastructure is in a constant state of flux.

For instance, a collaboration can start with several homogeneous nodes that could be trained optimally with All-Reduce. If new participants bring devices with less bandwidth, it may be more efficient to use the original nodes as parameter servers. As more peers join, these servers will eventually become unable to handle the network load and the collaboration will need to switch to a different strategy.

Running efficient training on this kind of infrastructure requires a protocol that can dynamically assign roles to every peer given their hardware and network capabilities:

- **Compute performance:** Each peer $i \in 1, \ldots, n$ can compute gradients over s_i samples per second. A peer that is unable to compute gradients (i.e. that has no GPU) will have s_i=0.
- **Bandwidth:** Peers communicate with a limited throughput: d_i for download and u_i for upload.
- **Geographical limitations:** In addition to individual bandwidth, the communication throughput between two peers i, j is also restricted by t_{ij} and t_{ji} in each direction.

Given these constraints, our objective is to find a communication strategy that has the highest training throughput, that is, the one that *makes the most SGD steps with a target batch size B per unit of time*. In turn, the training throughput of a collaboration depends on how we split the load among the participants. Each peer can be assigned to compute gradients over a subset of training examples, aggregate a part of those gradients from all peers, or both.

For simplicity and efficiency, we use delayed parameter updates (DPU) [81] — a technique that allows gradient computation and communication to run in parallel, at the cost of exactly one round of staleness. This strategy can improve time to convergence for a wide range of models, including Transformers [81, 82]. That said, our approach can be easily adapted to non-concurrent updates.

With DPU, the frequency of training updates is determined by either the time to compute gradients or the time to aggregate them, whichever takes longer. In total, a collaboration processes $\sum_{i=1}^{n} s_i \cdot c_i$ samples per second, where c_i is the binary indicator denoting whether i-th peer is assigned to contribute gradients. Assuming the target batch size B, the frequency of the computation phase can be expressed as $F_{compute} = \sum_{i=1}^{n} s_i \cdot c_i \,/B$.

During the communication phase, each peer is first assigned to accumulate gradients over a fraction of model parameters. After that, everyone partitions their local gradients and sends each partition to the corresponding peer. On the other end, receiver nodes accumulate the gradients from all senders and return the average. In modern distributed training systems, this procedure is highly parallelized [34, 83]: a reducer can aggregate one chunk of gradients while downloading the next chunk and distributing the previous one back to the same senders.

In order to properly optimize the training throughput, we must account for this parallelism. As such, we explicitly define the speed a_{ij} at which peer i sends gradients to peer j for aggregation. In turn, j-th peer aggregates gradients from all peers at the rate of the slowest sender $a_j = \min_{i:c_i=1} a_{ij}$. The senders can then get the aggregated results from the j-th reducer at $g_{ji} \leq a_j$. Finally, the total a_{ij} and g_{ij} for each peer cannot exceed their maximum download/upload speed. The only exception is that transfer within one node ($a_{ii},\ g_{ii}$) does not count towards network throughput.

The frequency of the gradient aggregation phase is simply the rate at which the slowest peer can aggregate the full gradient vector: $F_{agg} = \min_i \sum_j g_{ji} \,/\, P$, where P is the number of model parameters. The final optimization problem can be formulated as follows:

$$
\begin{array}{lll}
\max\limits_{a,g,c} & \min\left(\frac{\sum_{i=1}^{n} s_i \cdot c_i}{B},\ \frac{\min_i \sum_j g_{ji}}{P}\right) & \\
\text{s.t.} & g_{ij} \leq \min_{k:c_k=1} a_{ki} & \forall i,j \\
 & \sum_{j \neq i} (a_{ji} + g_{ji}) \leq d_i & \forall i \\
 & \sum_{j \neq i} (a_{ij} + g_{ij}) \leq u_i & \forall i \\
 & a_{ij} + g_{ij} \leq t_{ij} & \forall i,j
\end{array}
\tag{1}
$$

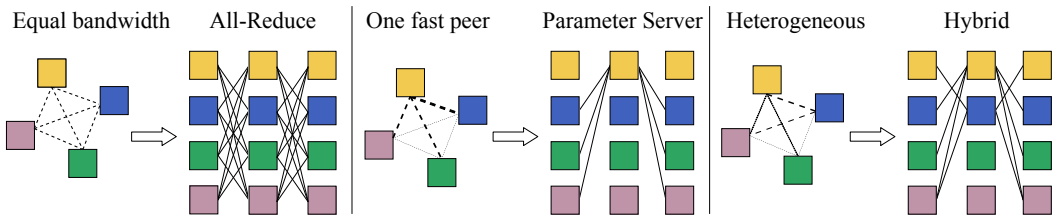


Figure 2: Example collaboration setups and corresponding strategies for optimal averaging. Each square represents one of the peers, line thickness denotes pairwise connection speed.

This problem must be solved regularly as participants are joining and leaving. Thus, we must ensure that the benefits of the optimal strategy outweigh the overhead of computing it. For that reason, we formulate optimal strategy search as a linear program that can be solved efficiently[3]. A more formal definition of problem (1) with detailed LP reduction can be found in Appendix B.

After this problem is solved, we assign each peer to aggregate a fraction of gradients proportional to $\min_j g_{ji}$. Peers with $c_i{=}1$ are also tasked with computing the gradients, while peers with $c_i{=}0$ remain idle and only participate in communication. This results in a natural division of labor. In the presence of many compute-heavy peers, some participants without accelerators will dedicate all their bandwidth to gradient aggregation instead of sending their local gradients.

Node failures. The resulting procedure can find the optimal communication strategy for averaging gradients across all participants. However, as the number of participants grows, it might be impractical to compute the global average due to node failures. Based on our experiments with several hundred active volunteers, most training iterations will have at least one participant with network issues. This implies that without necessary precautions, the entire averaging round will fail more often than it will succeed. To combat this issue, we use techniques [56, 29] that replace global averaging with several consecutive iterations in alternating groups of size m. The groups are chosen in such a way that the collaboration can obtain the exact average in $\log_m n$ steps. Furthermore, if any single participant fails, it will only affect his immediate group rather than the entire collaboration.

We adaptively choose the optimal group size m based on the number of peers and their failure rates. This optimization problem is independent of Equation (1) and aims to maximize the rate at which collaborators can compute the global average. We elaborate on this procedure in Appendix C.

Comparison with existing techniques. Our method was designed as a generalization of existing data-parallel strategies that recovers them in special cases. To illustrate this idea, we provide example configurations for which DeDLOC recovers specific well-known strategies:

1. **AR-SGD:** a homogeneous collaboration with reliable peers will use Butterfly All-Reduce [84];
2. **Parameter Server:** adding a single participant with a very high bandwidth and low compute performance will turn the previous collaboration into a parameter server [30];
3. **BytePS:** participants with the same bandwidth as AR-SGD nodes, but without compute accelerators, will behave as auxiliary summation services from BytePS [34];
4. **Decentralized SGD:** any collaboration with a sufficiently high failure rate will converge to $m{=}2$. In this mode, all communication is performed between pairs of nodes, similarly to D-PSGD [33].

However, when training with actual volunteer devices, DeDLOC typically follows a hybrid communication scheme that differs from each of the above options. We display several examples of schemes that can arise as a solution for the optimal strategy search problem in Figure 2.

3.3 System design

Training with volunteer hardware requires specialized system architecture that can dynamically scale with collaboration size and recover from node failures. DeDLOC achieves these properties by operating as a swarm, similarly in spirit to BitTorrent [85] and I2P [86]. Individual peers coordinate by forming a Distributed Hash Table — a fully decentralized fault-tolerant key-value storage [87, 88]. Collaborators use this shared "dictionary" to count the number of accumulated gradients, find groups for averaging and keep track of the training progress.

[3] In our experiments, the LP solver consistently converged in $<$ 50ms and was called $\approx$ 2 times per minute.

DeDLOC ensures that all peers use up-to-date parameters by tracking the number of global steps of each peer. If a peer skips a step, it will observe that others made more steps and download the latest parameters and optimizer statistics from one of the up-to-date peers before resuming training.

In order to ensure the integrity of DHT throughout the training run, DeDLOC requires a few peers with stable internet access. These "backbone" peers are responsible for welcoming new collaborators and performing auxiliary functions, such as storing checkpoints and tracking learning curves. The only requirement for those peers is that at least one of them is available at all times. As such, the backbone peers can be hosted on inexpensive servers without GPU (see Appendix F for cost analysis).

All other devices are treated as regular collaborators. Depending on their hardware and network bandwidth, these devices can be assigned to (i) compute gradients, (ii) aggregate gradients computed by other peers or (iii) do both, according to the adaptive averaging algorithm. However, performing these steps with actual volunteer devices requires solving another set of challenges described below.

Training under NAT and firewalls. In addition to having uneven compute and network capabilities, volunteer devices also deviate from traditional servers in network configuration. One major difference is the use of Network Address Translation (NAT) [89] — the technology that allows multiple devices to share the same IP address. In practice, the majority of household and organizational computers around the world use one or multiple layers of NAT (see Appendix D for more details). Unfortunately for distributed training, NAT makes it harder to establish peer-to-peer connections [90].

When operating under NAT, DeDLOC participants use one of the following techniques:

1. **Hole punching:** use a third peer to temporarily open access to both devices. Once both peers are accessible, they can establish a direct connection and transfer data as usual [90];
2. **Circuit relays:** both devices connect to a relay (another peer that is mutually accessible), then forward all communication through that relay [91];
3. **Client mode:** if everything else fails, a peer can still send gradients to others without the need for incoming connections. This imposes an additional constraint $a_i = 0$ for Equation (1).

A similar set of strategies can be found in a wide range of distributed systems that rely on peer-to-peer communication, such as WebRTC, VoIP (IP telephony), and BitTorrent. Most of these systems rely on dedicated servers to establish connections between peers. However, in our case it is more appealing to use a fully decentralized NAT traversal where the regular peers perform hole punching and relaying by themselves. We describe this approach in more detail in Appendix E.

Training on large datasets. Many prospective applications of DeDLOC require training on large datasets that can take multiple hours to download. We circumvent this problem by allowing participants to download the data progressively during training. To support this behavior, we split the dataset into shards; upon joining the collaboration, a peer begins downloading examples shard by shard in a streaming fashion. Once the first several examples are obtained, a collaborator can begin training right away while downloading the rest of data in background.

To ensure that the training examples are independent and identically distributed, each participant loads shards in a different random order and uses a buffer to shuffle the data within each shard. Each participant loads the first $S = 10,000$ examples into a buffer, then randomly picks a training batch from this buffer and replaces the chosen examples with newly downloaded ones. In our experiments, we stream the training data from a dedicated storage service. However, this service can be replaced with a peer-to-peer data sharing protocol akin to BitTorrent; see Appendix H for details.

Collaborator authentication. Many prospective applications of DeDLOC need a way to keep track of individual peer contributions and protect against malicious peers. In our experiments, we achieve this using an allowlist authentication system that we describe in Appendix I.5.

4 Experiments

In this section, we evaluate the performance of DeDLOC in realistic collaborative training conditions. Our primary focus is on training models that are useful for a wide range of downstream tasks and thus would attract a large number of collaborators. One area that fits this description is self-supervised learning, i.e., learning reusable feature representations on large unlabeled datasets. First, we conduct controlled experiments on two popular self-supervised learning tasks in Sections 4.1 and 4.2. Then, we set up a real-world collaborative training run with volunteers and report our findings in Section 4.3.

4.1 Self-supervised learning of visual representations

Our first set of experiments uses SwAV [92] — a self-supervised learning technique that learns image representations by contrasting cluster assignments. Similarly to the original paper, we train the ResNet-50 [93] model on the ImageNet dataset [1] without labels. Our experiments follow the recommended training configuration [92, 94]: 2+6 random crops, early prototype freezing and a queue with 3,840 samples for each worker, LARS [78] optimizer, and 32,768 samples per batch across all workers. In this and further experiments, we use Hivemind [95] to implement the infrastructure for decentralized averaging. We train with three hardware setups: SERVER, WORKSTATION and HYBRID. The SERVER setup contains 8 workers, each with a single V100 GPU and 1 Gb/s symmetric bandwidth. In turn, the WORKSTATION setup consists of 16 nodes with 1080 Ti and 200 Mb/s bandwidth per worker. Finally, the HYBRID setup combines both previous configurations for a total of 24 nodes. Unlike servers, workstation GPUs train in full precision because they do not support accelerated float16 computations [96].

We report learning curves for each hardware configuration in Figure 3. As expected, the HYBRID setup converges the fastest, beating SERVER and WORKSTATION setups by 40% and 52% accordingly. When used in a supervised setting (Section 4.1 from the original paper), the model learned in this setup achieves a comparable accuracy of 72.2%. Another important observation is that the workstation-only experiment achieves reasonable training throughput despite using dated hardware. To provide more insight into the performance of DeDLOC, we also measure the time it takes to run averaging in different configurations. We report the mean over 100 averaging rounds; the standard deviation was below 1% in all setups. As demonstrated in Table 1, adaptive averaging does not affect the performance for homogeneous setups while running 1.9 times faster on the hybrid infrastructure.

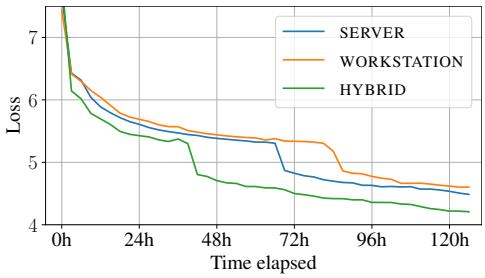


Figure 3: SwAV pretraining performance.

Setup	Algorithm		
	AR	PS	Ours
A: 8x1Gb/s	**1.19**	4.73	1.20
B: 16x0.2Gb/s	**5.3**	39.6	**5.3**
C: A + B	5.69	14.1	**2.96**
D: B + 1x2.5Gb/s	5.3	3.22	**3.18**

Table 1: ResNet-50 averaging performance.

4.2 Self-supervised pretraining for language understanding

Next, we investigate how collaborative training performs for more complex models. In this experiment, we pretrain the ALBERT-large [7] masked language model on the WikiText-103 dataset [97]. We chose this setup for two reasons: first, ALBERT is very sensitive to the choice of hyperparameters, and specifically batch size, even more than regular Transformers [75]. This makes it easier to verify that DeDLOC can reproduce the training conditions of regular data-parallel training. Second, because of weight sharing, training ALBERT is relatively more compute- and less communication-intensive than regular BERT [6], which makes it possible to train with lower bandwidth.

As before, we follow the exact training configuration from the original paper, but use GPUs instead of TPUs. We use the implementation of ALBERT from the `transformers` library [99]. We run all experiments on cloud instances with Tesla T4 GPUs and report the training loss as a function of time, similarly to [17, 38]. In order to evaluate how DeDLOC performs with different network speeds, we consider the following setups on the same platform with controlled conditions:

- **High-bandwidth:** 16 workers, each with Tesla T4 and 25 Gb/s symmetric bandwidth;
- **Heterogeneous:** same, but with 4x 200 Mb/s, 8x 100 Mb/s and 4x 50 Mb/s bandwidths;
- **Heterogeneous + load balancing:** like Heterogeneous, but with adaptive averaging (Section 3.2);
- **Auxiliary peers:** the previous setup with 4 additional CPU-only peers at 1 Gb/s bandwidth.
- **Time-varying:** same as previous, but with 8 additional peers at 100 Mb/s. The extra peers are training part-time, jointly alternating between 8 hours of training and 8 hours of downtime.

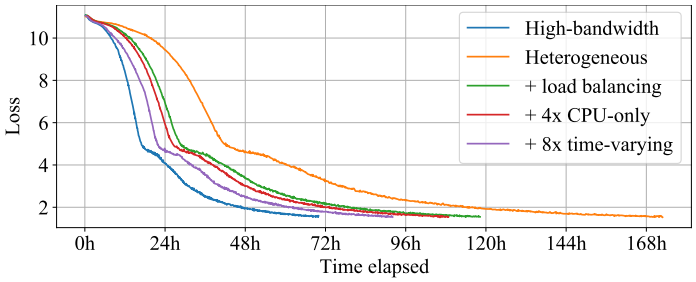


Figure 4: ALBERT pretraining performance.

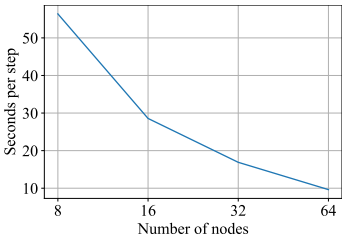


Figure 5: Scalability measurements for ALBERT pretraining.

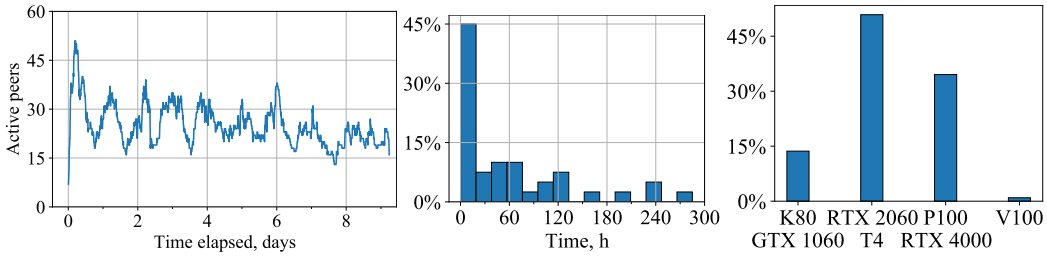
As one can see in Figure 4, naïve training with low-bandwidth peers results in an $\approx$ 2.5x slowdown compared to high-bandwidth ones. Enabling load balancing accelerates that setup by $\approx 47\%$. This effect grows to over 60% when adding 4 auxiliary peers. Finally, adding 8 part-time peers allows the collaboration to train at 74% the speed of the high-bandwidth setup without sacrificing the training stability. This turns the latter setup into a viable alternative to traditional distributed training without the need for expensive infrastructure (see the cost analysis in Appendix F). In addition, we demonstrate the high scalability of DeDLOC in Figure 5, which was obtained by running the same experiment with a varying number of nodes and measuring the time between gradient descent steps.

4.3 Real-world collaborative training

For our final evaluation, we organized an actual collaborative training run with volunteer participants, who were asked to pretrain a Transformer masked language model for the Bengali language. This task was chosen deliberately to show the benefits of collaborative training: Bengali has over 230M native speakers who can benefit from recent advances in NLP, but there are few pretrained models available for this language. We recruited 30 Bengali-speaking volunteers and 10 outside collaborators. All participants received instructions for contributing with free cloud platforms and access to the code for training on local computers. To avoid bias, we did not encourage any specific form of participation: volunteers were free to choose what hardware they contributed and for how long.

Specifically, we trained the ALBERT-large model on Wikipedia and the Bengali part of the OSCAR [100] multilingual corpus. The model was named sahajBERT after conducting a poll among the participants. We adapted our preprocessing by following the best practices for the Bengali language described in Appendix I.3. To stream from a mix of Wikipedia and OSCAR, the training process iteratively sampled examples from one or the other dataset, as described in Section 3.3. We accounted for uneven size and quality of data by oversampling Wikipedia by a factor of 2, which resulted in mixing probabilities of 0.23 for Wikipedia and 0.77 for OSCAR. Other hyperparameters were set to the same values as in Section 4.2. Also, in Appendix I.7 we report the results of sahajBERT-XL — a four times larger model with a specialized architecture that used both GPU and TPU resources.

In total, the 40 volunteers contributed compute time from 91 unique devices, most of which were running episodically. Figure 6b shows that although the median GPU time contributed by volunteers across all devices was $\approx$ 1.5 days, some participants ran the training script on several devices, attaining more than 200 hours over the duration of the experiment. With the exception of the start and the end of the collaborative run, the number of simultaneously active devices mostly varied between 15 and 35 depending on the local time. There was less activity in the last 3 days, likely because the volunteers could see that the model has converged on a public Weights & Biases [101] dashboard.



(a) Collaboration activity. (b) Participation time histogram. (c) Summary of volunteer hardware with example GPUs.

Figure 6: Collaborative experiment summary.

As depicted in Figure 6c, individual device performance varied significantly among the collaborators. Along with the resources provided by participants, we also used 16 preemptible single-GPU cloud T4 instances for training. We have estimated that the average volunteer device consumed 6.95 GB of network traffic per hour of training. While this bandwidth usage is by no means insignificant, it is comparable with cloud gaming [102] or high-quality video streaming [103].

The model converged after 8 days of training, which is 1.8x as fast as regular distributed training with 8 V100 GPUs that we ran as a baseline; Figure 7 displays the convergence plots for both setups. At the same time, the stepwise learning curves of the two runs were virtually identical (see Appendix I.6), which supports our hypothesis that training with DeDLOC is equivalent to a regular large-batch SGD.

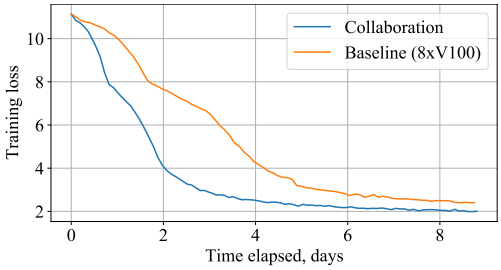


Figure 7: Training progress of sahajBERT.

Model	Wikiann F1	NCC Accuracy
bnRoBERTa	82.32 ± 0.67	80.94 ± 0.45
IndicBERT	92.52 ± 0.45	74.46 ± 1.91
XLM-R	96.48 ± 0.22	90.05 ± 0.38
sahajBERT	95.45 ± 0.53	91.97 ± 0.47
sahajBERT-XL	$\mathbf{96.59 \pm 0.26}$	$\mathbf{92.91 \pm 0.43}$

Table 2: Downstream evaluation results.

Finally, we compared the Bengali language representations of sahajBERT with those of other pretrained models on several downstream applications. The first model is XLM-R Large [9] — a cross-lingual Transformer-based masked language model that was pretrained on 100 languages and remains a strong baseline for multilingual representation learning. Similarly to sahajBERT, the second model, IndicBERT [104], is also based on the ALBERT architecture; however, it was pretrained on 12 languages, including Bengali and Indian English. The third model, bnRoBERTa [105], is a RoBERTa architecture trained on a monolingual Bengali corpus. We evaluate the model quality on two tasks: WikiANN [106] named entity recognition dataset and Soham News Category Classification benchmark from IndicGLUE [104]. For a detailed description of the setup, refer to Appendix I.8.

As shown in Table 2, sahajBERT performs comparably to three strong baselines despite being pretrained in a heterogeneous and highly unstable setting. Notably, our collaboratively trained model outperforms two specialized monolingual baselines and demonstrates competitive results to XLM-R Large, even though the latter has significantly more parameters (560 million instead of 17 million) and was trained on five hundred high-performance data center GPUs instead of tens of low-cost or even free-tier accelerators. This result confirms previous findings on the benefits of parameter sharing that were made by authors of ALBERT. Also, it highlights one additional advantage of such architectures: specifically, one can train a high-quality representation model in a communication-constrained setting (for instance, over the Internet) without facing noticeable data transfer bottlenecks.

5 Conclusion

In this work, we proposed DeDLOC — a collaborative deep learning approach that enables large-scale collective distributed training on whichever computers available to participants, regardless of hardware and network limitations. We demonstrated with several experiments that this is a viable approach that maintains its efficiency in a broad range of conditions. Finally, we report the first real collaborative training run of such a scale and share our findings on volunteer activity to pave the road for similar experiments in the future.

An essential property of collaborative training is its environmental impact. While all distributed training experiments have a negative impact due to carbon emissions [107], DeDLOC has one unique advantage. Due to the ability to utilize heterogeneous low-end devices, it can prolong the effective lifespan of existing computers. We discuss other aspects of environmental impact in Appendix J.

One issue that needs to be addressed before starting collaborative experiments is the need to gather a community of volunteers. Although our proposed authentication mechanism (see Appendix I.5) allows acknowledging participants for their contributions (briefly discussed in Appendix I.2), the best approach to recruit volunteers is an open question: one needs to take into account both the resources of community members and their motivation for training a specific model.

Acknowledgements

We thank Stas Bekman, Dmitry Abulkhanov, Roman Zhytar, Alexander Ploshkin, Vsevolod Plokhotnyuk and Roman Kail for their invaluable help with building the training infrastructure. Also, we thank Abhishek Thakur for helping with downstream evaluation and Tanmoy Sarkar with Omar Sanseviero, who helped us organize the collaborative experiment and gave regular status updates to the participants over the course of the training run. Finally, we thank the anonymous reviewers for their feedback on the content and the presentation of our paper.

In addition, authors would like to thank the students of Yandex School of Data Analysis who volunteered to participate in preliminary experiments.

We kindly thank all participants of the Neuropark community[4] who contributed to sahajBERT training. Below, we list the community members who agreed to provide their name for this paper: Aakash Gupta, Aninda Goswamy, Anjali Prasad, Anurag Singh, Arijit Sarkar, Chirranjit Ghosh, Debajit Mallick, Ibraheem Muhammad Moosa, Ishan Bagchi, Khalid Saifullah, Laxya Agarwal, Manan Dey, Mir Ali, Mrinal Mathur, Nilavya Das, Preetha Suri, Priyadarshan Sarkar, Sagnik Roy, Sahil Saha, Sanjeev Kumar, Sanskar Upadhyay, Shyam Sunder Kumar, Soumi Kaibartya, Subhranil Sarkar, Sujit Pal, Syed Modassir Ali, Tanmoy Sarkar, and Vaishali Pal.

Training sahajBERT-XL and hybrid GPU-TPU experiments were made possible by John Kintree, Debajit Mallick, Avijit Saha, Ishan Bagchi, Nilavya Das, Priyadarshan Sarkar, Sagnik Roy, Eduard Pokonechnyy, Arina Ruck. Finally, we would like to acknowledge Tanmoy Sarkar for setting up the backbone peer for sahajBERT-XL on his server and contributing to the evaluation codebase.

The computational resources for internal experiments on cloud instances were provided by the Amazon Research Awards program.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: a large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 06 2009.

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[3] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[4] J. Donahue, Y. Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*, pages 647–655, 2014.

[5] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, volume 9906, pages 694–711, 10 2016.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 06 2019.

[7] Zhen-Zhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[4] huggingface.co/neuropark

[9] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, 07 2020.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

[11] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using GPU model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[12] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Advances in Neural Information Processing Systems*, volume 33, pages 12449–12460, 2020.

[13] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

[14] Amy X. Lu, Haoran Zhang, Marzyeh Ghassemi, and Alan Moses. Self-supervised contrastive learning of protein representations by mutual information maximization. *bioRxiv*, 2020.

[15] Shion Honda, Shoi Shi, and H. Ueda. SMILES transformer: Pre-trained molecular fingerprint for low data drug discovery. *arXiv preprint arXiv:1911.04738*, 2019.

[16] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. *arXiv preprint arXiv:2104.04473*, 2021.

[17] Jiahuang Lin, Xin Li, and Gennady Pekhimenko. Multi-node BERT-pretraining: Cost-efficient approach. *arXiv preprint arXiv:2008.00177*, 2020.

[18] TensorFlow Hub. https://www.tensorflow.org/hub. Accessed: 2021-05-20.

[19] PyTorch Hub. https://pytorch.org/hub/. Accessed: 2021-05-20.

[20] Hugging Face Hub. https://huggingface.co/models. Accessed: 2021-05-20.

[21] Ryan Chard, Zhuozhao Li, Kyle Chard, Logan Ward, Yadu Babuji, Anna Woodard, Steven Tuecke, Ben Blaiszik, Michael J. Franklin, and Ian Foster. DLHub: Model and data serving for science. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 283–292, 05 2019.

[22] Pratik Joshi, Sebastin Santy, Amar Budhiraja, Kalika Bali, and Monojit Choudhury. The state and fate of linguistic diversity and inclusion in the NLP world. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6282–6293, 07 2020.

[23] David Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: An experiment in public-resource computing. *Commun. ACM*, 45:56–61, 11 2002.

[24] A. L. Beberg, D. Ensign, G. Jayachandran, S. Khaliq, and V. Pande. Folding@home: Lessons from eight years of volunteer distributed computing. *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8, 2009.

[25] David P Anderson. BOINC: A system for public-resource computing and storage. In *Fifth IEEE/ACM international workshop on grid computing*, pages 4–10. IEEE, 2004.

[26] Folding@home gets 1.5+ exaflops to fight covid-19. https://blogs.nvidia.com/blog/2020/04/01/foldingathome-exaflop-coronavirus/. Accessed: 2021-05-20.

[27] C. Tapparello, Colin Funai, Shurouq Hijazi, Abner Aquino, Bora Karaoglu, H. Ba, J. Shi, and W. Heinzelman. Volunteer computing on mobile devices: State of the art and future research directions. In *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, pages 153–181, 2016.

[28] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69:117–124, 02 2009.

[29] Shigang Li, Tal Ben-Nun, Giorgi Nadiradze, Salvatore Digirolamo, Nikoli Dryden, Dan Alistarh, and Torsten Hoefler. Breaking (global) barriers in parallel stochastic optimization with wait-avoiding group averaging. *IEEE Transactions on Parallel and Distributed Systems*, page 1–1, 2020.

[30] Mu Li, D. Andersen, J. Park, Alex Smola, Amr Ahmed, V. Josifovski, J. Long, E. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 2014 International Conference on Big Data Science and Computing*, 2014.

[31] Shaohuai Shi, Qiang Wang, and Xiaowen Chu. Performance modeling and evaluation of distributed deep learning frameworks on GPUs. In *IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, pages 949–957, 2018.

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

[33] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[34] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 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)*, pages 463–479, 11 2020.

[35] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, pages 103–112, 2019.

[36] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[37] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimization towards training a trillion parameter models. In *SC*, 2020.

[38] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[39] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep Gradient Compression: Reducing the communication bandwidth for distributed training. In *The International Conference on Learning Representations*, 2018.

[40] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, volume 23, pages 2595–2603, 2010.

[41] Sebastian Urban Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

[42] Anastasia Koloskova, Tao Lin, Sebastian U Stich, and Martin Jaggi. Decentralized deep learning with arbitrary communication compression. In *International Conference on Learning Representations*, 2020.

[43] Zhize Li, Dmitry Kovalev, Xun Qian, and Peter Richtarik. Acceleration for compressed gradient descent in distributed and federated optimization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5895–5904, 07 2020.

[44] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[45] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, Broomfield, CO, October 2014. USENIX Association.

[46] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1223–1231, 2012.

[47] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[48] Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U-chupala, Yoshiki Tanaka, and Yuichi Kageyama. Massively distributed SGD: ImageNet/ResNet-50 training in a flash. *arXiv preprint arXiv:1811.05233*, 2019.

[49] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, 2020.

[50] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, 02 2005.

[51] Paul Sack and William Gropp. Collective algorithms for multiported torus networks. *ACM Trans. Parallel Comput.*, 1(2), February 2015.

[52] PyTorch Elastic. https://pytorch.org/elastic. Accessed: 2021-05-20.

[53] Elastic Horovod. https://horovod.rtfd.io/en/stable/elastic_include.html. Accessed: 2021-05-20.

[54] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. Stochastic gradient push for distributed deep learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 344–353, 06 2019.

[55] Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael Rabbat. SlowMo: Improving communication-efficient distributed SGD with slow momentum. In *International Conference on Learning Representations*, 2020.

[56] Max Ryabinin, Eduard Gorbunov, Vsevolod Plokhotnyuk, and Gennady Pekhimenko. Moshpit SGD: Communication-efficient decentralized training on heterogeneous unreliable devices. *arXiv preprint arXiv:2103.03239*, 2021.

[57] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[58] K. A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé M Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *Proceedings of Machine Learning and Systems (MLSys)*, 2019.

[59] Thorsten Wittkopp and Alexander Acker. Decentralized federated learning preserves model and data privacy. In *International Conference on Service-Oriented Computing*, pages 176–187. Springer, 2020.

[60] Stefan M. Larson, Christopher D. Snow, Michael Shirts, and Vijay S. Pande. Folding@home and Genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *arXiv preprint arXiv:0901.0866*, 2009.

[61] Folding@home update on SARS-CoV-2 (10 mar 2020). foldingathome.org/2020/03/10/covid19-update. Accessed: 2021-05-20.

[62] Javier Barranco, Yunhi Cai, David Cameron, Matthew Crouch, Riccardo De Maria, Laurence Field, M. Giovannozzi, Pascal Hermes, Nils Høimyr, Dobrin Kaltchev, Nikos Karastathis, Cinzia Luzzi, Ewen Maclean, Eric Mcintosh, Alessio Mereghetti, James Molson, Yuri

Nosochkov, Tatiana Pieloni, Ivan Reid, and Igor Zacharov. LHC@Home: a BOINC-based volunteer computing infrastructure for physics studies at CERN. *Open Engineering*, 7, 12 2017.

[63] Jeongnim Kim, Andrew D Baczewski, Todd D Beaudet, Anouar Benali, M Chandler Bennett, Mark A Berrill, Nick S Blunt, Edgar Josué Landinez Borda, Michele Casula, David M Ceperley, Simone Chiesa, Bryan K Clark, Raymond C Clay, Kris T Delaney, Mark Dewing, Kenneth P Esler, Hongxia Hao, Olle Heinonen, Paul R C Kent, Jaron T Krogel, Ilkka Kylänpää, Ying Wai Li, M Graham Lopez, Ye Luo, Fionn D Malone, Richard M Martin, Amrita Mathuriya, Jeremy McMinis, Cody A Melton, Lubos Mitas, Miguel A Morales, Eric Neuscamman, William D Parker, Sergio D Pineda Flores, Nichols A Romero, Brenda M Rubenstein, Jacqueline A R Shea, Hyeondeok Shin, Luke Shulenburger, Andreas F Tillack, Joshua P Townsend, Norm M Tubman, Brett Van Der Goetz, Jordan E Vincent, D ChangMo Yang, Yubo Yang, Shuai Zhang, and Luning Zhao. QMCPACK: an open sourceab initioquantum monte carlo package for the electronic structure of atoms, molecules and solids. *Journal of Physics: Condensed Matter*, 30(19):195901, 04 2018.

[64] Folding@home project timeline. https://foldingathome.org/project-timeline. Accessed: 2021-05-20.

[65] B. Steltner, M. A. Papa, H. B. Eggenstein, B. Allen, V. Dergachev, R. Prix, B. Machenschalk, S. Walsh, S. J. Zhu, and S. Kwang. Einstein@Home all-sky search for continuous gravitational waves in LIGO O2 public data. *The Astrophysical Journal*, 909(1):79, 03 2021.

[66] Michael Gross. Folding research recruits unconventional help. *Current biology : CB*, 22:R35–8, 01 2012.

[67] Tetsu Narumi, Shun Kameoka, Makoto Taiji, and Kenji Yasuoka. Accelerating molecular dynamics simulations on playstation 3 platform using virtual-grape programming model. *SIAM J. Scientific Computing*, 30:3108–3125, 01 2008.

[68] John Clemens. MLDS: A dataset for weight-space analysis of neural networks. *arXiv preprint arXiv:2104.10555*, 2021.

[69] Gian-Carlo Pascutto and Gary Linscott. Leela chess zero. lczero.org, 2019. Accessed: 2021-05-20.

[70] Ekasit Kijsipongse, Apivadee Piyatumrong, and Suriya U-ruekolan. A hybrid gpu cluster and volunteer computing platform for scalable deep learning. *The Journal of Supercomputing*, 04 2018.

[71] Medha Atre, Birendra Jha, and Ashwini Rao. Distributed deep learning using volunteer computing-like paradigm. *arXiv preprint arXiv:2103.08894*, 2021.

[72] Max Ryabinin and Anton Gusev. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. In *Advances in Neural Information Processing Systems*, volume 33, pages 3659–3672, 2020.

[73] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, page 2672–2680, 2014.

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008, 2017.

[75] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110, 03 2018.

[76] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763, 11 2020.

[77] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, page 1–15, 2019.

[78] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[79] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[80] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, 10 2018.

[81] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-offload: Democratizing billion-scale model training. *arXiv preprint arXiv:2101.06840*, 2021.

[82] Alham Fikri Aji and Kenneth Heafield. Making asynchronous stochastic gradient descent work for transformers. *Proceedings of the 3rd Workshop on Neural Generation and Translation*, 2019.

[83] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, August 2020.

[84] Zhenyu Li, James Davis, and Stephen Jarvis. An efficient task-based all-reduce for machine learning applications. In *Proceedings of the Machine Learning on HPC Environments*, MLHPC'17, New York, NY, USA, 2017. Association for Computing Machinery.

[85] Bram Cohen. The BitTorrent Protocol Specification. http://www.bittorrent.org/beps/bep_0003.html, 2008.

[86] jrandom (Pseudonym). Invisible internet project (i2p) project overview. geti2p.net/_static/pdf/i2p_philosophy.pdf, August 2003. Accessed: 2021-05-20.

[87] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[88] M Frans Kaashoek and David R Karger. Koorde: A simple degree-optimal distributed hash table. In *International Workshop on Peer-to-Peer Systems*, pages 98–107. Springer, 2003.

[89] Andrew Biggadike, Daniel Ferullo, Geoffrey Wilson, and Adrian Perrig. NATBLASTER: Establishing TCP connections between hosts behind NATs. In *Proceedings of ACM SIGCOMM ASIA Workshop*, 2005.

[90] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, page 13, USA, 2005. USENIX Association.

[91] T. Reddy, A. Johnston, P. Matthews, and J. Rosenberg. Traversal using relays around NAT (TURN): Relay extensions to session traversal utilities for NAT (STUN). RFC 8656, 02 2020.

[92] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems*, volume 33, pages 9912–9924, 2020.

[93] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[94] Priya Goyal, Quentin Duval, Jeremy Reizenstein, Matthew Leavitt, Min Xu, Benjamin Lefaudeux, Mannat Singh, Vinicius Reis, Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Ishan Misra. Vissl. https://github.com/facebookresearch/vissl, 2021.

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

[96] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018.

[97] Stephen Merity, Caiming Xiong, James Bradbury, and R. Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2017.

[98] 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. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[99] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 10 2020.

[100] Pedro Javier Ortiz Suárez, Laurent Romary, and Benoît Sagot. A monolingual approach to contextualized word embeddings for mid-resource languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1703–1714, 07 2020.

[101] Lukas Biewald. Experiment tracking with Weights and Biases, 2020. Software available from wandb.com.

[102] Google Stadia data usage. https://support.google.com/stadia/answer/9607891. Accessed: 2021-05-20.

[103] Netflix data usage. https://help.netflix.com/en/node/87. Accessed: 2021-05-20.

[104] Divyanshu Kakwani, Anoop Kunchukuttan, Satish Golla, Gokul N.C., Avik Bhattacharyya, Mitesh M. Khapra, and Pratyush Kumar. IndicNLPSuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for Indian languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4948–4961, 11 2020.

[105] Kushal Jain, Adwait Deshpande, Kumar Shridhar, Felix Laumann, and Ayushman Dash. Indic-transformers: An analysis of transformer language models for Indian languages. *arXiv preprint arXiv:2011.02323*, 2020.

[106] Xiaoman Pan, Boliang Zhang, Jonathan May, Joel Nothman, Kevin Knight, and Heng Ji. Cross-lingual name tagging and linking for 282 languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1946–1958, 07 2017.

[107] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems, 07 2020.

[108] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[109] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[110] Seymour Kaplan. Application of programs with maximin objective functions to problems of optimal resource allocation. *Operations Research*, 22(4):802–807, 1974.

[111] Erling D. Andersen and Knud D. Andersen. The MOSEK interior point optimizer for linear programming: An implementation of the homogeneous algorithm. In *Applied Optimization*, pages 197–232. Springer US, 2000.

[112] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, Ilhan Polat, Yu Feng, Eric W. Moore, J. Vanderplas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Daniel Henriksen, E. A. Quintero, Charles R. Harris, A. M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa,

Paul van Mulbregt, Aditya Alessandro Pietro Alex Andreas Andreas Anthony Ant Vijaykumar Bardelli Rothberg Hilboll Kloeckner Sco, Aditya Vijaykumar, Alessandro Pietro Bardelli, Alex Rothberg, Andreas Hilboll, Andre Kloeckner, Anthony M. Scopatz, Antony Lee, Ariel S. Rokem, C. Nathan Woods, Chad Fulton, Charles Masson, Christian Häggström, Clark Fitzgerald, David A. Nicholson, David R. Hagen, Dmitrii V. Pasechnik, Emanuele Olivetti, Eric A. Martin, E. Wieser, Fabrice Silva, Felix Lenders, Florian Wilhelm, Gert Young, Gavin A. Price, Gert-Ludwig Ingold, Gregory E. Allen, Gregory R. Lee, Hervé Audren, Irvin Probst, Jorg P. Dietrich, Jacob Silterra, James T. Webber, Janko Slavic, Joel Nothman, Johannes Buchner, Johannes Kulick, Johannes L. Schönberger, José Vinícius de Miranda Cardoso, Joscha Reimer, Joseph E. Harrington, Juan Luis Cano Rodríguez, Juan Nunez-Iglesias, Justin Kuczynski, Kevin Lee Tritz, Martin Dr Thoma, Matt Newville, Matthias Kümmerer, Maximilian Bolingbroke, Michael Tartre, Mikhail Pak, Nathaniel J. Smith, Nikolai Nowaczyk, Nikolay Shebanov, Oleksandr Pavlyk, Per Andreas Brodtkorb, Perry Lee, Robert T. McGibbon, Roman Feldbauer, Sam Lewis, Sam Tygier, Scott Sievert, Sebastiano Vigna, Stefan Peterson, Surhud More, Tadeusz Pudlik, Taku Oshima, Thomas J. Pingel, Thomas P. Robitaille, Thomas Spura, Thouis Raymond Jones, Tim Cera, Tim Leslie, Tiziano Zito, Tom Krauss, U. Upadhyay, Yaroslav O. Halchenko, and Y. Vázquez-Baeza. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261 – 272, 2020.

[113] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs). RFC 3489, 03 2003.

[114] libp2p. https://libp2p.io/. Accessed: 2021-05-20.

[115] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[116] Sebastian U. Stich. Unified optimal analysis of the (stochastic) gradient method, 2019.

[117] Ahmed Khaled, Othmane Sebbouh, Nicolas Loizou, Robert M. Gower, and Peter Richtárik. Unified analysis of stochastic gradient methods for composite convex and smooth optimization, 2020.

[118] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 66–75, 2018.

[119] Anthony MOI, Pierric Cistac, Nicolas Patry, Evan P. Walsh, Funtowicz Morgan, Sebastian Pütz, Thomas Wolf, Sylvain Gugger, Clément Delangue, Julien Chaumond, Lysandre Debut, and Patrick von Platen. Hugging face tokenizers library. https://github.com/huggingface/tokenizers, 2019.

[120] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. Datasets: A community library for natural language processing. *arXiv preprint arxiv:2109.02846*, 2021.

[121] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[122] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[123] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[124] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[125] Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[126] Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Févry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam M. Shazeer, Zhenzhong Lan, Yanqi Zhou, Wen hong Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

[127] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[128] Stella Biderman, Sid Black, Charles Foster, Leo Gao, Eric Hallahan, Horace He, Ben Wang, and Phil Wang. Rotary embeddings: A relative revolution. blog.eleuther.ai/rotary-embeddings, 2021. Accessed: 2021-05-20.

[129] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2Net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 11 2015.

[130] Afshin Rahimi, Yuan Li, and Trevor Cohn. Massively multilingual transfer for NER. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 151–164, 07 2019.

[131] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[132] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[133] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 3645–3650, 2019.

[134] Roy Schwartz, Jesse Dodge, Noah Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63:54–63, 2020.

[135] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43, 2020.

[136] Donald Kline, Nikolas Parshook, Xiaoyu Ge, E. Brunvand, R. Melhem, Panos K. Chrysanthis, and A. Jones. Holistically evaluating the environmental impacts in modern computing systems. *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, 2016.

[137] Rabih Bashroush. A comprehensive reasoning framework for hardware refresh in data centers. *IEEE Transactions on Sustainable Computing*, 3:209–220, 2018.

[138] Xinchi Qiu, Titouan Parcollet, Daniel J. Beutel, Taner Topal, Akhil Mathur, and Nicholas D. Lane. Can federated learning save the planet? *arXiv preprint arXiv:2010.06537*, 2020.