Distributed Deep Learning In Open Collaborations

Anonymous Author(s)
Affiliation
Address
email

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 nearly 50 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, 13], reinforcement learning[14], and computational biology [15, 16].

Training these models is a notoriously difficult and time-consuming task: it often requires hundreds of high-end GPU servers [10, 17] and would take multiple years on a single device [18]. 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 [19, 20, 21, 22]. 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 [23]. In turn, computer vision hubs rarely feature models trained on drawings, satellite images, 3D renders, microscopy or any other data that

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

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 [24, 25, 26]. The raw compute performance of such collaborations often exceeds that of the fastest supercomputers [27]; 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 [28]. 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 [29] works well for identical servers but suffers significant performance penalties from network latency or bandwidth variation [30]. Another technique known as Parameter Server can handle heterogeneous devices at the cost of being less scalable [31]. Applying any of these strategies outside their preferred conditions may significantly reduce the training throughput [32], 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 [33], All-Reduce SGD [34], decentralized SGD [35], BytePS [36], 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[1].

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 [37, 38, 39], 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 [39, 17, 40].

Training on multiple nodes was first implemented with parameter server (PS) [33]. 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

[1]Code and training configurations are available at `github.com/neurips-submit/DeDLOC`

extensions to PS that alleviate the bottleneck by reducing the communication load [41, 42, 43, 44, 45], introducing asynchronous updates [46, 47] or training with multiple servers [48, 36].

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 [49, 50, 51]. All-Reduce is a family of collective operations that allow nodes to efficiently aggregate (e.g. sum) their local vectors and distribute the result across all devices [52, 53, 54]. 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 [55, 56], 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 are 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 [57, 58] or running All-Reduce in small alternating groups [59, 30]. 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 [35, 30].

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 [60]. For that reason, federated learning algorithms prioritize data privacy over training efficiency, often leaving most of the compute resources unused [61, 62]. 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 idle time of their desktops, smartphones and other personal devices to collectively solve a computationally hard problem. This approach has seen successful applications in bioinformatics, physics and other scientific areas [63, 64, 65, 24, 66, 67, 68].

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 [27]. 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 [26]. Volunteer computing systems were also the first "supercomputers" to reach 1 petaFLOP and 1 exaFLOP barriers [27, 69]. These results became possible due to the contributions of a broad range of devices from high-end workstations to smartphones and even gaming consoles[70].

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 to 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 [71], 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 [72, 73, 74]. 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 [75], 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 [76] and Transformers [77], require a strict training regimen. Deviating from the recommended batch size or introducing stale gradients may significantly affect the training outcome [78, 79, 80]. 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 [81, 82], which allows peers to communicate less frequently. This strategy also provides a natural way to deal with heterogeneous hardware [83]: 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. Figure 1 gives a high-level visual explanation of this algorithm.

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.

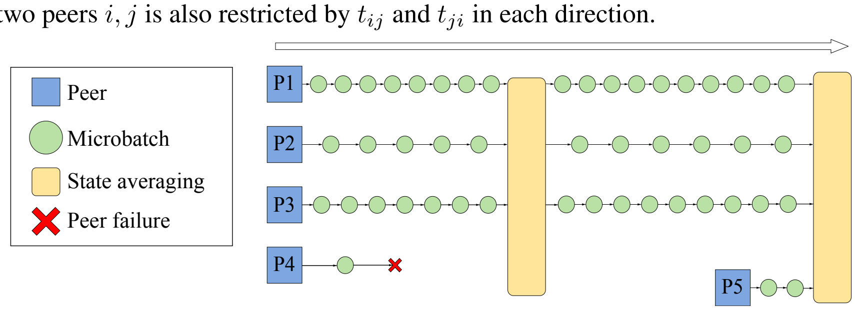


Figure 1: Two DeDLOC training iterations with example collaborator dynamics.

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) [84] — 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 [84, 85]. 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 [36, 86]: 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 peer 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 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:

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

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[2]. A more formal definition of problem (1) with the 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 [59, 30] 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.

[2]In our experiments, the LP solver consistently converges in < 50ms and is called ≈ 2 times per minute.

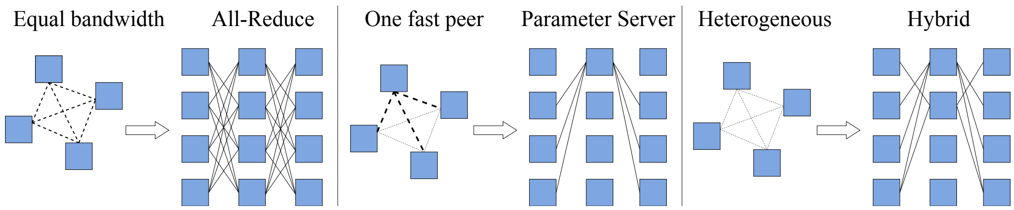


Figure 2: Example collaboration setups and corresponding strategies for optimal averaging.

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 [87];
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 [33];
3. **BytePS:** participants with the same bandwidth as AR-SGD nodes, but without compute accelerators, will behave as auxiliary summation services from BytePS [36];
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 [35].

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 [88] and I2P [89]. Individual peers coordinate by forming a Distributed Hash Table — a fully decentralized fault-tolerant key-value storage [90, 91]. Collaborators use this shared "dictionary" to count the number of accumulated gradients, find groups for averaging and keep track of the training progress.

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) [92] — 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 [93].

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 [94];
2. **Circuit relays:** both devices connect to a relay (another peer that is mutually accessible), then forward all communication through that relay [95];
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 G 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 H.4.

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 [96] — a self-supervised learning technique that learns image representations by contrasting cluster assignments. Similarly to the original paper, we train the ResNet-50 [97] model on the ImageNet dataset [1] without labels. Our experiments follow the recommended training configuration [96, 98]: 2+6 random crops, early prototype freezing and a queue with 3,840 samples for each worker, LARS [81] optimizer and 32,768 samples per batch across all workers. 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 float16 acceleration [99].

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. Another important observation is that the workstation-only experiment achieves a 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 Figure 1, adaptive averaging does not affect the performance for homogeneous setups but runs 1.9 times faster on the hybrid infrastructure.

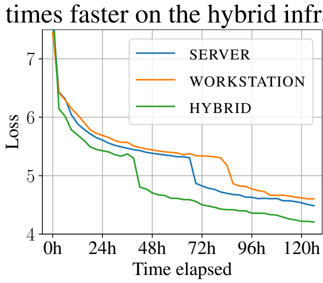


Figure 3: SwAV pretraining.

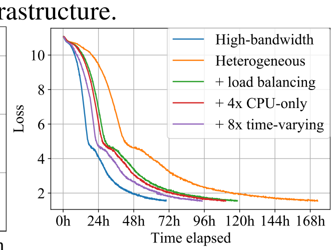


Figure 4: ALBERT pretraining performance.

Table 1: ResNet-50 averaging 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 with PS (1x2.5Gb/s)	5.3	3.22	**3.18**

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 [100]. We chose this setup for two reasons: first, ALBERT is very sensitive to the choice of hyperparameters and specifically batch size, even more so than regular transformers [78]. 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 [103]. We run all experiments on cloud instances with Tesla T4 GPUs and report the training loss as a function of time, similarly to [18, 40]. In order to evaluate how DeDLOC performs with different network speeds, we consider the following setups on the same platform with controlled conditions:

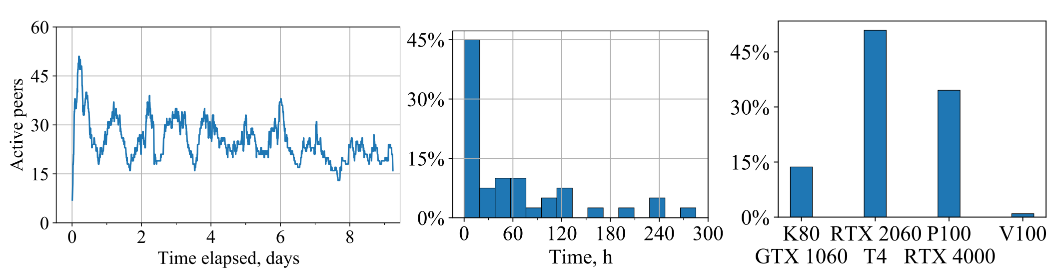
- **High-bandwidth:** 16 workers, each with Tesla T4 and 25 Gb/s symmetric bandwidth;
- **Heterogeneous:** same, but with 4x 200 Mb/s, 8x 100 Mb/s and 4x 50 Mb/s bandwidths;
- **Heterogeneous + load balancing:** like Heterogeneous, but with adaptive averaging (Section 3.2);
- **Auxiliary peers:** the 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.

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).

4.3 Real-world collaborative training

For our final evaluation, we organized an actual collaborative training run with volunteer participants. In this experiment, we asked collaborators to pretrain a Transformer [77] model for the Bengali language. This task was chosen deliberately to showcase the benefits of collaborative training: Bengali has over 230M native speakers that can benefit from recent advances in NLP, but there are few pretrained models available for this language. We recruited 38 Bengali-speaking volunteers and 11 outside collaborators. All participants received instructions for contributing with local computers and free cloud platforms. To avoid bias, we did not encourage any specific form of participation: volunteers were free to choose what hardware they contribute and for how long.

Specifically, we trained the ALBERT-large architecture on Wikipedia and the Bengali part of the OSCAR [104] 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 H.2. To stream from a mix of Wikipedia and OSCAR, the training process iteratively samples 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.



(a) Collaboration activity. (b) Distribution of participation time. (c) Summary of volunteer hardware with examples.

Figure 5: Collaborative experiment summary.

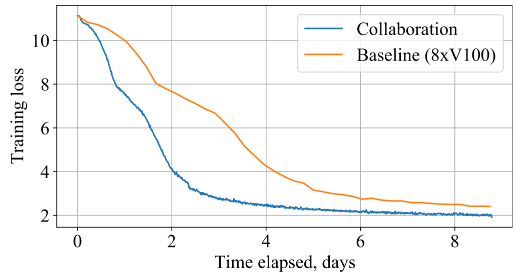


Figure 6: Training progress of sahajBERT.

Table 2: Downstream task performance of pretrained models on Bengali language benchmarks.

Model	Wikiann F1	NCC Accuracy
sahajBERT	95.45 ± 0.53	**91.97 ± 0.47**
XLM-R	**96.48 ± 0.22**	90.05 ± 0.38
IndicBERT	92.52 ± 0.45	74.46 ± 1.91
bnRoBERTa	82.32 ± 0.67	80.94 ± 0.45

In total, the 49 volunteers contributed compute time from 91 unique devices, most of which were running episodically. Figure 5b 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 [105] dashboard.

As depicted in Figure 5c, 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. Regarding the network utilization, we have estimated that the average volunteer device consumed 6.95 GB of network traffic per hour of training. While this bandwidth usage by no means insignificant, it is comparable with cloud gaming [106] or high-quality video streaming [107].

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 (see Figure 6). At the same time, the stepwise learning curves of the two runs were virtually identical, which supports our hypothesis that training with DeDLOC is equivalent to a regular large-batch SGD.

In addition, we compare the Bengali language representations of sahajBERT with other pretrained models on several downstream tasks. The first model is XLM-R Large [9] — a Transformer network pretrained on 100 languages, which remains a strong baseline for multilingual representation learning. The second model, IndicBERT [108], is also based on the ALBERT architecture and pretrained on 12 languages including Bengali and Indian English. The third model, bnRoBERTa [109], is a RoBERTa architecture trained on monolingual Bengali. We evaluate the model quality on two downstream tasks in Bengali: Wikiann [110] named entity recognition dataset and Soham News Category Classification benchmark from IndicGLUE [108]. As shown in Table 2, sahajBERT performs comparably to three recent strong baselines despite being pretrained in a heterogeneous and highly unstable setting. For more details regarding the downstream evaluation, refer to Appendix H.6.

5 Conclusion & Broader Impact

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 scale and share our findings on volunteer activity to pave the road for similar experiments in the future.

An important aspect of collaborative training is its environmental impact. While all distributed training experiments have negative impact due to carbon emissions [111], DeDLOC has one unique advantage. Due to its ability to utilize low-end heterogeneous devices, collaborative training can prolong the effective lifespan of existing computers, reducing the waste from hardware overhaul. We discuss this in Appendix I.

One issue that needs to be addressed before starting collaborative experiments is the need to gather a community of volunteers. DeDLOC is equally suitable for artificial "communities" composed of inexpensive preemptible cloud instances, existing groups of people or communities created specifically for the experiment (as described in Section 4.3). Although our proposed authentication mechanism (see Appendix H.4) allows to acknowledge participants for their contribution, the development of the best approach to recruit volunteers is an open question: one needs to take into account both the available resources of community members and their motivation for training a specific model.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 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*, CVPR '14, page 580–587, USA, 2014. IEEE Computer Society.

[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 *ICML*, 2014.

[5] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 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*, abs/1907.11692, 2019.

[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, Online, July 2020. Association for Computational Linguistics.

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 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, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020.

[13] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12449–12460. Curran Associates, Inc., 2020.

[14] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey, 2021.

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

[16] Shion Honda, Shoi Shi, and Hiroki R. Ueda. Smiles transformer: Pre-trained molecular fingerprint for low data drug discovery, 2019.

[17] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters. *arXiv preprint arXiv:2104.04473*, 2021.

[18] Jiahuang Lin, Xin Li, and Gennady Pekhimenko. Multi-node bert-pretraining: Cost-efficient approach, 2020.

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

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

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

[22] R. Chard, Z. Li, K. Chard, L. Ward, Y. Babuji, A. Woodard, S. Tuecke, B. Blaiszik, M. J. Franklin, and I. Foster. Dlhub: Model and data serving for science. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 283–292, Los Alamitos, CA, USA, may 2019. IEEE Computer Society.

[23] Pratik M. Joshi, Sebastin Santy, A. Budhiraja, K. Bali, and M. Choudhury. The state and fate of linguistic diversity and inclusion in the nlp world. *ArXiv*, abs/2004.09095, 2020.

[24] 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.

[25] 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.

[26] 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.

[27] *Folding@home gets 1.5+ Exaflops to Fight COVID-19*. Accessed: 2021-05-20.

[28] 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.

[29] 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.

[30] 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.

[31] 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 *BigDataScience '14*, 2014.

[32] Shaohuai Shi, Qiang Wang, and Xiaowen Chu. Performance modeling and evaluation of distributed deep learning frameworks on gpus, 2018.

[33] Mu Li. Scaling distributed machine learning with the parameter server. In *Proceedings of the 2014 International Conference on Big Data Science and Computing*, BigDataScience '14, New York, NY, USA, 2014. Association for Computing Machinery.

[34] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow, 2018.

[35] 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 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[36] 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. USENIX Association, November 2020.

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

[38] 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.

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

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

[41] 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.

[42] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23, pages 2595–2603. Curran Associates, Inc., 2010.

[43] Sebastian Urban Stich. Local SGD converges fast and communicates little. *International Conference on Learning Representations (ICLR)*, page arXiv:1805.09767, 2019.

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

[45] Zhize Li, Dmitry Kovalev, Xun Qian, and Peter Richtarik. Acceleration for compressed gradient descent in distributed and federated optimization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5895–5904. PMLR, 13–18 Jul 2020.

[46] 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.

[47] 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.

[48] 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 F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1223–1231. Curran Associates, Inc., 2012.

[49] 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, 2017.

[50] Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U-chupala, Yoshiki Tanaka, and Yuichi Kageyama. Massively distributed sgd: Imagenet/resnet-50 training in a flash, 2019.

[51] 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.

[52] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distrib. Comput.*, 69(2):117–124, February 2009.

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

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

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

[56] Elastic Horovod. `https://horovod.readthedocs.io/en/stable/elastic_include.html`. Accessed: 2021-05-20.

[57] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. Stochastic gradient push for distributed deep learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 344–353. PMLR, 09–15 Jun 2019.

[58] 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.

[59] Max Ryabinin, Eduard Gorbunov, Vsevolod Plokhotnyuk, and Gennady Pekhimenko. Moshpit sgd: Communication-efficient decentralized training on heterogeneous unreliable devices, 2021.

[60] 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.

[61] 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 *SysML 2019*, 2019. To appear.

[62] Thorsten Wittkopp and Alexander Acker. Decentralized federated learning preserves model and data privacy, 2021.

[63] Stefan Larson, Christopher Snow, Michael Shirts, and Vijay Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *arXiv*, 02 2009.

[64] *Folding@home update on SARS-COV-2 (10 MAR 2020)*. Accessed: 2021-05-20.

[65] 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.

[66] 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, apr 2018.

[67] *Folding@home project timeline*. Accessed: 2021-05-20.

[68] B. Steltner, M. A. Papa, H.-B. Eggenstein, B. Allen, V. Dergachev, R. Prix, B. Machenschalk, S. Walsh, S. J. Zhu, O. Behnke, and et al. Einstein@home all-sky search for continuous gravitational waves in ligo o2 public data. *The Astrophysical Journal*, 909(1):79, Mar 2021.

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

[70] 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.

[71] John Clemens. Mlds: A dataset for weight-space analysis of neural networks, 2021.

[72] Pascutto, Gian-Carlo and Linscott, Gary. Leela chess zero, 2019.

[73] 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.

[74] Medha Atre, Birendra Jha, and Ashwini Rao. Distributed deep learning using volunteer computing-like paradigm, 2021.

[75] Max Ryabinin and Anton Gusev. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3659–3672. Curran Associates, Inc., 2020.

[76] 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*, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

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

[79] 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, Online, November 2020. Association for Computational Linguistics.

[80] 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*, SOSP '19, page 1–15, New York, NY, USA, 2019. Association for Computing Machinery.

[81] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks, 2017.

[82] 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, 2020.

[83] 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, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[84] 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, 2021.

[85] Alham Fikri Aji and Kenneth Heafield. Making asynchronous stochastic gradient descent work for transformers, 2019.

[86] 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.

[87] 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.

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

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

[90] 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.

[91] 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.

[92] Andrew Biggadike, Daniel Ferullo, Geoffrey Wilson, and Adrian Perrig. Natblaster: Establishing tcp connections between hosts behind nats. In *IN PROCEEDINGS OF ACM SIGCOMM ASIA WORKSHOP*, 2005.

[93] B. Ford, P. Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference, General Track*, 2005.

[94] 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.

[95] Tirumaleswar Reddy.K, Alan Johnston, Philip Matthews, and Jonathan Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 8656, February 2020.

[96] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9912–9924. Curran Associates, Inc., 2020.

[97] 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.

[98] 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.

[99] 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.

[100] Stephen Merity, Caiming Xiong, James Bradbury, and R. Socher. Pointer sentinel mixture models. *ArXiv*, abs/1609.07843, 2017.

[101] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[102] Thomas Wolf, Quentin Lhoest, Patrick von Platen, Yacine Jernite, Mariama Drame, Julien Plu, Julien Chaumond, Clement Delangue, Clara Ma, Abhishek Thakur, Suraj Patil, Joe Davison, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angie McMillan-Major, Simon Brandeis, Sylvain Gugger, François Lagunas, Lysandre Debut, Morgan Funtowicz, Anthony Moi, Sasha Rush, Philipp Schmidd, Pierric Cistac, Victor Muštar, Jeff Boudier, and Anna Tordjmann. Datasets. *GitHub. Note: https://github.com/huggingface/datasets*, 1, 2020.

[103] 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, Online, October 2020. Association for Computational Linguistics.

[104] 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, Online, July 2020. Association for Computational Linguistics.

[105] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

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

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

[108] 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, Online, November 2020. Association for Computational Linguistics.

[109] Kushal Jain, Adwait Deshpande, Kumar Shridhar, Felix Laumann, and Ayushman Dash. Indic-transformers: An analysis of transformer language models for indian languages, 2020.

[110] 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 (Volume 1: Long Papers)*, pages 1946–1958, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[111] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *ArXiv*, abs/2007.03051, 2020.

[112] 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.

[113] 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. *CoRR*, abs/1911.00222, 2019.

[114] 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 *SysML 2019*, 2019. To appear.

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

[116] 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.

[117] 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, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[118] J. Weinberger, C. Huitema, and R. Mahy. Stun—simple traversal of user datagram protocol (udp) through network address translators (nats). *IETF RFC 3489*, 01 2003.

[119] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. *CoRR*, abs/cs/0603074, 2006.

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

[121] 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, Dandelion Mané, 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 Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[122] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*, 2018.

[123] 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.

[124] 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, Online, July 2020. Association for Computational Linguistics.

[125] 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, Florence, Italy, July 2019. Association for Computational Linguistics.

[126] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

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

[128] Emma Strubell, Ananya Ganesh, and A. McCallum. Energy and policy considerations for deep learning in nlp. *ArXiv*, abs/1906.02243, 2019.

[129] Roy Schwartz, Jesse Dodge, N. A. Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63:54 – 63, 2020.

[130] Peter Henderson, Jie-Ru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *ArXiv*, abs/2002.05651, 2020.

[131] 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.

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

[133] Xinchi Qiu, Titouan Parcollet, Daniel J. Beutel, Taner Topal, Akhil Mathur, and N. Lane. Can federated learning save the planet. *arXiv: Learning*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] First, in Section 4.1, we show that DeDLOC is outperformed by regular All-Reduce in case of a high-bandwidth network, which is often a part of an HPC cluster. Second, in Section 4.2 we discuss that models with a high number of parameters involve more data transfer for distributed gradient aggregation, and thus are less amenable to training in collaborative conditions with typical Internet connection speeds.
   (c) Did you discuss any potential negative societal impacts of your work? [No] The work describes a general approach for collaborative training of deep learning models that is independent of possible applications.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] The work does not include any theoretical statements besides the optimization problem formulation in Equation 1.
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code and instructions are available at `github.com/neurips-submit/DeDLOC`.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] For most of the experiments, we pretrain on all available data. We use standard setups, which are refered to in their respective sections. For downstream evaluation of sahajBERT, we specify the details in Appendix H.6.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Unfortunately, most of the experiments (especially the real-world volunteer training run) require a considerable amount of resources and could not be ran several times. However, for the SwAV averaging experiment, we ran the procedure 100 times in all configurations and observed no significant deviations from the mean. Also, in downstream evaluation of sahajBERT, we averaged the results after finetuning the models with 3 different random seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We describe our hardware setups in all experiments.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include our code for the experiments as a URL.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We use openly available datasets from prior work that were collected from the Internet, and the nature of these datasets has already been discussed before.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [Yes] See Section H.1.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] The participants contributed on a volunteer basis.

Supplementary Material

A Federated learning

Federated learning (FL) is an approach that trains the model on decentralized data stored on many devices without sharing private training data [60]. This scenario is currently gaining more popularity with the rising awareness of data privacy and emerging legal constraints, such as GDPR. Similarly to our setting, FL systems must deal with unreliable heterogeneous hardware. However, their main goal is to ensure the data privacy, which often leads to sacrifices in terms of efficiency.

Most practical FL systems utilize a central parameter server that aggregates local gradients from workers and updates the global model. As we increase the number of workers, the total system performance becomes bounded by the throughput of this server. The problem is exacerbated by secure aggregation protocols [112, 113] that further increase the communication overhead to ensure data privacy. To account for these limitations, production FL systems perform each update using only a small random subset of peers, while the rest remain idle [114]. Contrary to this, our goal is to maximize the training performance by running computations on all peers.

Another recent line of work explores federated learning algorithms with a decentralized communication topology. Maintaining data privacy in these conditions also requires specialized techniques that introduce communication overhead. For instance, [62] proposes a system where workers cannot share parameters directly, relying on a secure peer-to-peer knowledge distillation instead.

The above discussion makes it clear that the purpose of the federated learning is orthogonal to ours: we aim to train the global model on publicly available data and achieve the best possible performance.

B Optimal averaging strategy via linear programming

Recall that DeDLOC finds the optimal communication strategy by solving the following problem:

$$\begin{array}{lll}\max\limits_{a,g,c} & \min\left(\frac{\sum_{i=1}^{n} s_i \cdot c_i}{B},\ \frac{\min_i \sum_j g_{ji}}{P}\right) & \\ \text{s.t.} & g_{ij} \leq \min_{k: c_k = 1} a_{ki} & \forall i, j \\ & \sum_{j \neq i} \left(a_{ji} + g_{ji}\right) \leq d_i & \forall i \\ & \sum_{j \neq i} \left(a_{ij} + g_{ij}\right) \leq u_i & \forall i \\ & a_{ij} + g_{ij} \leq t_{ij} & \forall i, j \\ & a_{ij} \geq 0 \ \ \& \ \ g_{ij} \geq 0 \ \ \& \ \ c_i. \in \{0, 1\} & \forall i, j \end{array} \tag{2}$$

Here, a_{ij} denotes the fraction of network throughput allocated to sending gradients from peer i to peer j for aggregation, g_{ji} is the corresponding fraction for returning the averaged tensors back to sender, and c_i is a binary indicator that represents whether or not peer i computes gradients. The remaining variables are parameters that denote peer compute performance s_i, maximum download and upload speeds (d_i and u_i respectively) and regional limitations of peer-to-peer throughput ($t_i j$). Finally, B denotes the global target batch size per step and P is the number of model parameters.

As stated earlier in Section 3.2, the DeDLOC peers need to find the optimal strategy during each averaging round. As such, we must ensure that the procedure for solving (2) does not introduce any significant overhead. To that end, we reformulate the problem as a linear program by means of several consecutive reductions, which are described below.

Max-min LP reduction. First, we replace the original max-min objective with a linear one by following the technique described in [115]: we maximize a new surrogate variable ξ and replace the inner $\min$ by two additional constraints:

$$\begin{array}{lll}\max\limits_{a,g,c} & \xi & \\ \text{s.t.} & \xi \leq \frac{\sum_{i=1}^{n} s_i \cdot c_i}{B} & \\ & \xi \leq \frac{\sum_j g_{ji}}{P} & \forall i \end{array} \tag{3}$$

Binary to LP relaxation. Second, we must account for the binary variable c_i. From a formal perspective, using these indicators transforms our problem into a binary mixed-integer program with a combinatorial worst-case complexity. However, for this specific problem, it is possible to rewrite the constraints in such a way that c_i can be treated as a continuous variable $0 \le c_i \le 1$:

$$\forall i, j, k \in 1 \dots n \quad g_{ij} \le a_{ki} + (1 - c_k) \cdot d_i \tag{4}$$

For $c_k = 1$, the above equation (4) is exactly equivalent to the original constraint $g_{ij} \le \min_{k:c_k=1} a_{ki}$. In turn, setting $c_k < 1$ for some k effectively removes the corresponding peer k from the $\min$ operator, allowing participant i to aggregate tensors with up to its maximum download speed d_i instead of waiting for peer k. The d_i factor in (4) can be replaced with any large positive number as long as the constraint (4) is not saturated for c_k=0. In practice, $c_k \ne 1$ corresponds to peer k **not** computing gradients, but still assisting in gradient aggregation.

Applying the two above reductions, we get the following linear program:

$$\begin{array}{lll}
\max\limits_{a,g,c} & \xi & \\
\text{s.t.} & \xi \le \sum_{i=1}^{n} s_i \cdot c_i \;/\; B & \\
& \xi \le \sum_j g_{ji} \;/\; P & \forall i \\
& g_{ij} \le a_{ki} + (1 - c_k) \cdot d_i & \forall i, j, k \\
& \sum_{j \ne i} (a_{ji} + g_{ji}) \le d_i & \forall i \\
& \sum_{j \ne i} (a_{ij} + g_{ij}) \le u_i & \forall i \\
& a_{ij} + g_{ij} \le t_{ij} & \forall i, j \\
& a_{ij} \ge 0 & \forall i, j \\
& g_{ij} \ge 0 & \forall i, j \\
& 0 \le c_i \le 1 & \forall i
\end{array} \tag{5}$$

To avoid additional synchronization steps, each peer within DeDLOC solves the above problem (5) independently using the interior point solver [116]. Based on the obtained solution, peer i will aggregate a fraction of gradients proportional to its effective throughput:

$$\text{fraction}_i \propto \frac{\min_j g_{ij}}{\sum_k \min_j g_{kj}}. \tag{6}$$

Furthermore, if $c_i \ne 1$, the corresponding participant will disregard its local gradients. In the future, it may be possible to allow such peers to contribute partial gradients akin to [41]. However, we leave this investigation to future work.

For certain collaboration compositions, there can be multiple optimal strategies with equal training throughputs. To ensure that all participants act according to the same strategy, we require each peer to solve (5) using a deterministic interior point algorithm with globally consistent hyperparameters [117].

Another practical consideration is that some peers are unable to compute gradients or perform aggregation (for instance, due to networking issues described in Section 3.3). To account for these limitations, we exclude such peers from aggregation in $\frac{\sum_{i=1}^{n} s_i \cdot c_i}{B}$ and $\frac{\sum_j g_{ji}}{P}$ terms for compute and network resources respectively.

C Fault tolerance

In practice, using DeDLOC with large collaborations will eventually require dealing with node failures. If the failures are rare, it is possible restart the failed steps until they succeed. However, if the collaboration size increases, this strategy will eventually become impractical.

One possible solution is to replace the global (collaboration-wide) All-Reduce with several parallel operations, which is known as Group All-Reduce [30] or Moshpit All-Reduce [59]. Each operation involves a small independent group of m peers, whereas the groups themselves are formed in such a way that the collaboration can obtain the global average in a logarithmic number of rounds.

Under this strategy, any failed device will only affect its local group instead of the entire collaboration. Furthermore, each individual group will have a higher success rate, since it contains $m \ll n$ peers.

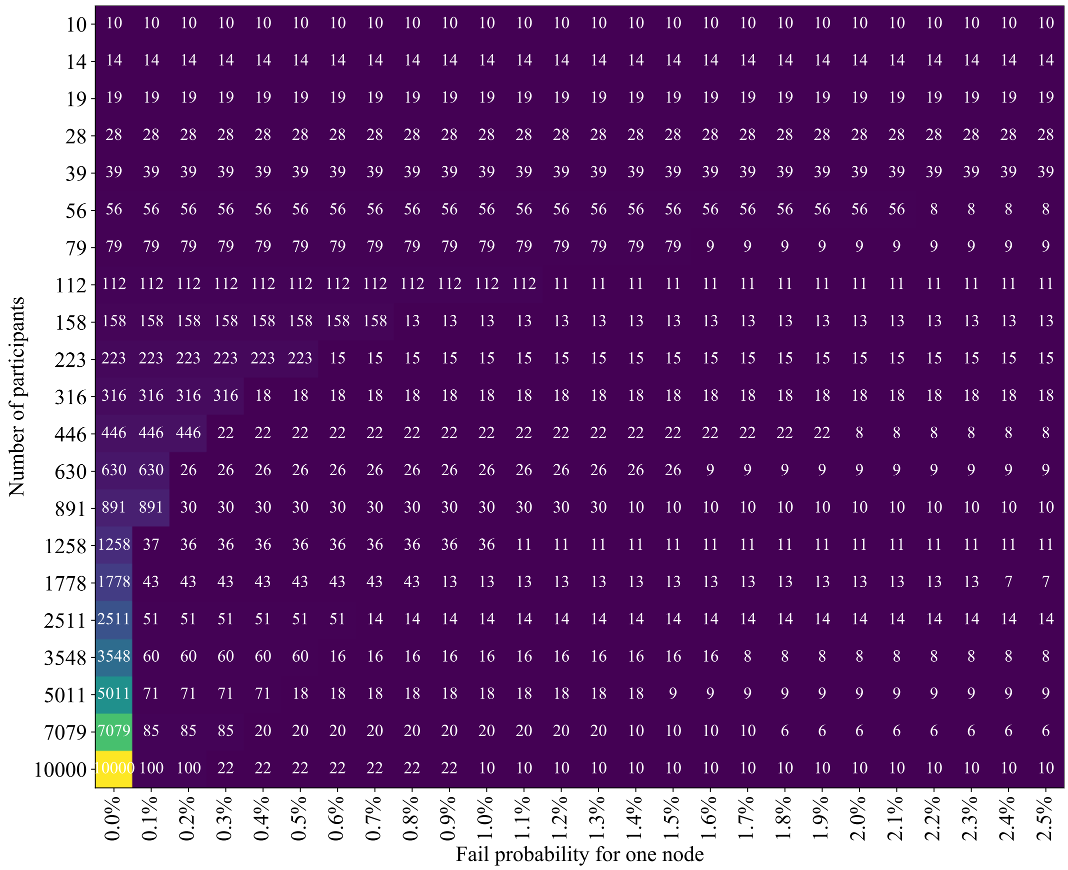


Figure 7: Optimal group size for different collaboration sizes and failure rates.

In turn, the drawback of using group-based All-Reduce is that the collaboration will need $\lceil \log_m n \rceil$ steps to obtain the global average.

We can select the optimal group size by minimizing the *expected* number of iterations required to compute the global average, including both restarts from node failures and the overhead from using Group All-Reduce. For reference, we include the optimal group sizes for typical collaborations and failure rates in Figure 7. In all our experiments, the optimal group size was $m{=}n$ due to a small number of participants and very rare significant network failures.

D Network address translation

Collaborative training, similarly to any other application incorporating peer-to-peer communication, is susceptible to a number of networking issues, among which the most common is the inability to accept incoming connections due to Network Address Translation, or NAT [92]. The primary function of NAT is to separate the address space of the local network from the global address space by dynamically translating addresses and port numbers of outgoing sessions into public endpoints. Therefore, NAT helps deter the rapid depletion of IPv4 addresses and provides additional security by hiding the local network structure from external parties. However, this also means that NAT devices only authorize outgoing connections, since the dynamic mapping of local endpoints makes it impossible to forward incoming packets to the proper internal host.

For the purposes of the current work, NAT devices can be categorized into two groups — cone and symmetric. A cone NAT translates an internal IP address and port to the same globally routable endpoint regardless of the destination host, whereas a symmetric NAT allocates different address mapping for each destination host. In case of UDP traffic, the cone NAT can be traversed using the mechanism of UDP Hole Punching. Briefly put, this technique consists of two stages. During the first phase, peers A and B connect to the same globally accessible rendezvous server using the STUN protocol [118] and exchange their public and private endpoints. The rendezvous server is often called the STUN server by the name of the protocol. At the next step, both peers start sending UDP data packets to each other's endpoints. If A's packet reaches NAT B before B's packet "punches a hole", then it is dropped by the NAT B, but when the B's packet reaches NAT A shortly after this, the outgoing session has already been initiated by A, so the B's request is successfully forwarded to A. If both peers happen to "punch a hole" in their NATs before the arrival of the counterpart's packet, then the connection is established immediately.

For the TCP traffic, hole punching is also possible, though it has to overcome additional API issues that arise because of the client-server paradigm around which TCP was designed. However, peer-to-peer communication over TCP connections is more robust than over UDP, since NAT usually timeouts the UDP port mapping, thus periodical keep-alive messages must be transmitted. As reported in [119], currently almost two thirds of all NAT vendors provide devices which are compatible with TCP hole punching, that is, consistently map private endpoints and do not send back Reset packets to unsolicited requests.

As for the symmetric NAT, only relaying through a third-party proxy can help establish the connection between peers. This is supported with the TURN protocol [95]. If two peers fail to connect via hole punching, they appeal to the TURN server for an interaction through it.

E Peer-to-peer network infrastructure

To enable peer-to-peer interactions that can bypass NAT, we can use the libp2p framework [120]. Each peer has a set of multiaddresses that allow other participants to establish a connection. Multiaddress comprises an IP address, an L4 protocol (TCP/UDP) with a port, an optional high-level protocol (QUIC), and a peer identifier. A peer can listen to several transport protocols, but it may have only one identifier.

After peers connect to the network, they can interact with each other via their respective identifiers. There are no dedicated STUN and TURN servers in the libp2p network: their role is played by public participants. The network must contain at least 4 publicly accessible peers to be able to recognize public addresses of newly connected peers. Optimally, these are well-known peers with multiaddresses known to all participants. Upon joining, a new node synchronizes with the DHT used for routing and receives information about other available peers. After that, a peer can interact with other participants using their peer id. If the network can get the public address of the peer, then other participants will be able to connect to it.

If a public address of the peer is not available or two peers are using different transport, the communication can be started by relaying requests via an intermediate participant. Libp2p supports the autorelay feature that allows finding the best relay automatically. When autorelay is enabled, a public peer can serve as a relay for other participants, and a private peer will find the best relay.

F Cost analysis

In this section, we provide a detailed cost analysis of several hardware and networking setups that can be used for both tasks described in Section 4, namely, SwAV and ALBERT pretraining.

For simplicity, we only consider temporary resource ownership, i.e., renting GPU-enabled servers instead of building it on-premise. The latter option can be more cost-efficient in the long term, but might be impractical if only a few training runs are required. For the same reason, we do not consider discounts available for committed usage of the same resource over multiple years.

As for the rented resources, there are several general hardware categories that we consider:

1. High-performance cloud GPU — dedicated instances with multiple high-end compute accelerators and extremely fast device interconnect.

2. Low-end cloud GPU — single-GPU instances with NVIDIA M60, T4 or P40, linked with a fast (preferably intra-datacenter) network of 10–50 Gb/s.

3. Commodity GPUs — regular desktop-like machines with consumer-grade GPUs, like NVIDIA RTX 2070, 2080 Ti, 3070. On average, they can have higher performance than low-end cloud devices, but lower network throughput (50–200 Mb/s).
4. Volunteer hardware — almost the same class of devices as in the previous section, with the same advantages and disadvantages, but "free" for the experiment organizers.

For a fair comparison, we consider three types of GPU instances: cloud V100, cloud T4 and commodity GPUs from peer-to-peer marketplaces, such as `vast.ai` or `golem.ai`. While several cloud providers offer newer generation GPUs (NVIDIA Ampere), this GPU lineup is still in an active rollout phase, which causes significant price fluctuations. Thus, we base our conclusions on more established generations of GPUs.

In addition to GPU instances, DeDLOC can also benefit from non-GPU servers that act as auxiliary parameter aggregators. The only real requirement for such servers is high network bandwidth. As such, we consider additional resource types:

1. Premium cloud VMs — low-end instances from premium cloud providers. We consider instances with 2 cores, 16GB RAM and 25 Gb/s maximum bandwidth (symmetric).
2. Economy cloud VMs — similar cloud instances (or dedicated servers) from economy cloud providers. For this run, we consider instances with the same 2 cores / 16GB RAM, but only 300–1000 Mb/s symmetric bandwidth (depending on the provider).
3. Volunteer non-GPU devices — in theory, it is possible to run collaborative training entirely on volunteer devices with zero hardware expenses for the organizer. However, we omit this option as it trivializes our cost analysis.

On top of that, all cloud and marketplace instances can be rented in a guaranteed ("on-demand") or a non-guaranteed option. In the latter scenario, the resources are offered at a significant discount, but the resource provider can terminate such instances at any time.

Based on the available resource types and ownership models, we assemble six server fleets with approximately equal training performance in our two experimental setups. For convenience, we order these setups by how difficult they are to operate (easiest-first):

- Single high-end node — 8 x NVIDIA Tesla V100: easiest to operate, but the most expensive option.
- Preemptible high-end node has the same hardware but costs less due to irregular availability, which creates a need for regularly saved checkpoints.
- Distributed nodes — 16 x NVIDIA Tesla T4: homogeneous, require distributed optimization.
- Distributed + preemptible — same but preemptible, can be used with a framework that supports elastic training, such as TorchElastic[55] or Elastic Horovod[56].
- Distributed + heterogeneous — 5x NVIDIA GTX 1080 Ti, 3x RTX 2070, 1x 2070S, 2x 2080, 4x 2080 Ti, 1x 3070. This configuration has lower bandwidth, thus additional CPU-only peers are needed for efficient averaging.
- Collaborative training — for this setup, we assume that the GPUs from the previous setup are available from volunteers. In that case, the only sources of expenses for the organizer are networking and CPU-only nodes.

Table 3: Costs of training setups.

	Cloud on-demand		Cloud preemptible		Marketplace	Volunteer
Instance types	8xV100	16xT4	8xV100	16xT4	4xCPU+16xGPU	4xCPU
Monthly price	$16898	$5299	$5133	$2074	$5148	$257

As one can see in Table 3, using a single high-end node is the most expensive alternative. Switching to multiple lower-end nodes and using non-guaranteed instances reduces the cost by a factor of $\approx$ 3x each. Finally, the volunteer infrastructure is two orders of magnitude cheaper than the high-performance setup. However, some of this price difference is effectively shifted to volunteers. Based on average electricity and networking costs of household Internet connections, we estimate the expense at $9-30 *per volunteer per month*, assuming 16 volunteers with equivalent GPUs. However, actual costs can vary based on the region, time duration and the exact hardware used by each volunteer.

Finally, we want to reiterate that the above setups require different amounts of effort (and expertise). Training on a single high-end node can be done with virtually no code changes in major deep learning frameworks, such as TensorFlow [121] or PyTorch [101]. In contrast, multi-node (and especially elastic) setups require specialized distributed training frameworks and careful performance tuning. Finally, working with volunteer or marketplace instances introduces a new layer of complexity that we address in this paper.

Networking costs. When done naïvely, training with geographically distributed participants can incur significant networking expenses. For instance, when using preemptible cloud GPUs from a major provider, allocating these GPUs in different regions can incur additional costs of more than $3000 per month, compared to a total hardware cost of $2074 for the same period.

More importantly, using premium non-GPU instances for collaborative training will also incur additional networking costs. Based on our preliminary experiments, a collaborative training setup equivalent to Table 3 would lead to an average networking bill of $5000-6000 per month. Fortunately, it is possible to circumvent this expense by using cloud providers that do not charge additional costs for network traffic. These providers typically offer less reliable instances with lower maximum bandwidth, which is not a significant issue for DeDLOC.

As a general recipe for reproducing our experiments, we recommend using one of the two setups. When running experiments internally, one can use any major cloud provider as long as all instances are *configured to avoid cross-regional networking costs* (e.g. use internal address space). In contrast, when training with actual volunteer devices, we recommend using cloud providers without additional networking charges or existing server infrastructure.

G Decentralized data streaming

In this section, we propose a generalization of our data streaming approach described in Section 3.3 to a setting without any central data storage. Namely, we offer a way to to distribute large datasets across all participants by sharding the examples in the same manner that was used previously.

Specifically, this approach is based on the notion of a local buffer combined with the decentralized metadata storage enabled by the DHT. When a peer joins the experiment, the training process allocates a buffer for several chunks on a local high-capacity storage device (HDD/SSD) available to that peer; the number of chunks is determined by the participant and depends on the hardware capabilities of their computer. Then, in order to procure training data, the peer queries the DHT to find the shards that are stored on the least number of other peers. Assuming that the number of shards does not exceed several thousand, this search can be done by a simple linear-time lookup of all keys without any significant performance drawbacks. After finding such shards, the training process randomly chooses one shard from this set and downloads it from another peer. When the download is complete, the participating node trains on batches from this shard and stores it for later use by other members of the network. The training process repeats such iterations; if the local buffer becomes full at any point, the shards with the highest replication factor are evicted in favor of new data.

The decentralized approach to data streaming has two immediate benefits. First, similarly to distributed training, this approach reduces the load on a single server (or the content delivery network), which might result in significant savings for large-scale experiments that use datasets hosted by cloud providers. Second, even when the data is hosted by organizers of the collaborative experiment, its size might be too large to prevent efficient storage and sharing without investments in specialized infrastructure, which is often quite expensive as well. Storing small portions of the dataset on the computers of participants allows circumventing both issues by distributing the load among all peers. However, we note that the above approach was not implemented for our current experiments; this section is intended to serve as a description of future work.

H Collaborative experiment setup

H.1 Instructions for participants

All communication with volunteer contributors took place on a group instant messaging platform. Prior to launching the experiment itself, we used this platform to communicate with Bengali speakers in order to validate the language-specific elements of the model, such as the normalization component of the tokenizer and the sentence splitter tool.

Then, for the collaborative training, we first sent several introductory messages before the event to explain what the event will consist of. Then, we sent a message the day before and a message on the event's launch day with instructions on how to join the training run. Lastly, we sent daily messages to report the current status of the event. An anonymized version of the event's launch day message can be found in Figure 8. In this message, the volunteers were invited to:

1. Submit their account names on the digital identity provider we used for validation;
2. Once added to the allow-list, join the training via notebooks provided by the organizers. After checking that the connection was established and that the GPU was available, participants had to run the notebook and fill in the necessary credentials for the identity platform.

Hi @everyone! We're starting the Collaborative Training Experiment now! Here is some important information:

How to participate?
1. As a reminder, you need to provide your digital identity provider username to be able to participate. For the current participants, $name_1$ already gathered this list (thank you $name_1$!). For new participants, please join *#albert-allowlist* and add your username. Someone from the team will add you to the allowlist. If you see a ⏳ reaction, we're on it! If you see a ✅, you should be added by then. Feel free to reach out to $name_2$, $name_3$, $name_4$, $name_5$, $name_6$ or me if you don't have access.
2. You can join the training with:

- **Colab**: notebook access link
- **Kaggle**: notebook access link
  This option provides you a P100 and lasts longer than Colab. This requires a Kaggle account. You must **enable Internet access and switch kernel to GPU mode** explicitly. If it is stuck at "installing dependencies" for over 5 minutes, it means you changed the session type too late. Simply restart with GPU/Internet enabled and it should work just fine.

Please do not run multiple GPU instances on the same service! You can use Kaggle in one tab and Colab in another, but avoid having two Colab GPU instances at the same time.

Local run: if you have a local GPU and you're tech-savvy. We will keep you informed when this option is available. Stay tuned!

Feel free to ask any questions in *#albert-bengali-training* channel and reach out to us (at the right you can see the list of organizers).
In the following dashboard you can track the status of training: link

Thank you all for participating and let us know if you have any questions!

Figure 8: An anonymized instructions message sent at the event launch

H.2 Tokenizer

For this experiment, we used the architecture of the ALBERT model [7]; the authors of the original work have chosen the unigram language model [122] token segmentation algorithm that allows transforming a raw text into subword units based on a fixed size vocabulary of 30k tokens.
In order to use the tokenizer that is adapted to the Bengali language, we created a new tokenizer using the Tokenizers library [123].

This tokenizer is composed of:

- Several normalizations adapted to the Bengali language: NMT normalization, NFKC normalization, removal of multiple spaces, homogenization of some recurring unicode characters in the Bengali language and lowercasing;
- Specific pre-tokenization rules to condense the vocabulary: we split on whitespaces and replace them with an underscore character "▁" (U+2581), we also isolate all punctuation and digits from any other characters;
- A Unigram language model as a segmentation algorithm with a 32k tokens vocabulary, trained on the deduplicated Bengali subset of OSCAR [104];
- A template postprocessor, allowing a special token "[CLS]" to be included at the beginning of the sequence, as well as a special token "[SEP]" to separate a pair of segments and to denote the end of sequence.

H.3 Dataset streaming

Streaming the data to each participant allows to start training immediately, since the participants do not have to download the full dataset before launching the training. More specifically, the examples from the dataset can be downloaded progressively as training goes. To do so, we used the datasets library [102]. It enabled streaming of Wikipedia and OSCAR, as well as shuffling, on-the-fly processing and mixing of the datasets.

For the experiment, we use the Wikipedia and OSCAR Bengali datasets. Both datasets are split in shards, respectively in the Parquet and GZIP-compressed raw text formats. Information about the datasets is given in Table 4. The participants download the examples from those files during training, since it is possible to iterate row group by row group from Parquet files and line by line from compressed text files.

The Bengali Wikipedia dataset is based on the 03/20/2021 Wikipedia dump. The data was processed using the Wikipedia processing script of the datasets library in early April of 2021. Each example contains the content of one full article, cleaned from markup and sections such as references.

Table 4: Sizes of the Bengali Wikipedia and OSCAR datasets used for training.

	Wikipedia	OSCAR
Uncompressed size	657MB	6.2 GB
Documents	167,786	1,114,481
Shards	10	4

To shuffle the datasets, we make each participant iterate over the shards in random order. Then, a shuffle buffer of size $S = 10000$ is used, which is compatible with the progressive download of examples. We use a shuffle buffer, because we do not want the participants to download entire shards in the beginning of training just for shuffling.

Sentence splitting, tokenization and preprocessing for next sentence prediction are applied to the examples in an online manner. Since these steps are several orders of magnitude faster that forward and backward passes of the model, they have no significant impact on the training performance.

H.4 Participant authentication

Since our experiment was an open collaboration, we chose to set up an authentication system allowing only the people motivated by the final result of the model to join the training. Allow-listing seemed to be the most suitable solution to this need. We therefore distinguish between three types of actors in the distributed network:

- *Central server's moderators*: people who start the experiment, maintain the whitelist and know how to join the training. They have a pair $(public\_key_{auth}, private\_key_{auth})$ of public-private keys securely hosted on the central authentication server. In this protocol, the role of the central server is threefold: 1) to verify the identity of a collaborator requesting the confirmation of the identity provider website, 2) to verify that this collaborator is whitelisted and 3) to distribute access passes to authorized collaborators. Peers have a secure HTTPS-based communication channel with this server in order to protect the data;

- *Digital identity provider*: an entity which is able to create digital identities via a website. In order to create the allowlist, moderators asked collaborators to have a digital identity on an identity provider website. This has several advantages: the collaborators have the feeling of belonging to a community which is a great vector of enthusiasm and motivation, moderators can acknowledge each collaborators' contribution and it prevents bots from joining the training. In our setup, each identity linked to a username can be claimed by a login and a password owned by one collaborator;
- *Collaborators / Peers*: people who wish to make their computing resources available for the collaborative training. Each peer i in the network has a pair $(public\_key_i, private\_key_i)$ of public-private keys. They also have a digital identity on a identity provider website.

The following procedures aim to prevent 1) that a non-allow-listed collaborator can communicate with members of the collaborative training and 2) that a malicious actor could claim to be a allow-listed collaborator:

- *Joining the network*: To join the collaborative training, a peer i must request an access pass from the authorization server. To grant the access pass, the authorization server asks the digital identity provider if the peer is who he claims to be. If the entity provider confirms the identity of the peer, the authorization server checks that the username appears in the allow-list. If these two steps are verified, the authorization server creates an access pass otherwise it rejects the peer's request. The access pass is temporary and contains the following information:
  - the endpoint of a peer already present in the network
  - an access token $access\_token_i$ composed of a string containing the peer's username, its public key $public\_key_i$ and the expiration date of its access pass signed with the private key $private\_key_{auth}$.
  - the public key $public\_key_{auth}$

  With this access pass, the peer can make requests and responds to requests in the decentralized network. After expiration, the peer may repeat this procedure to get a new token.
- *Making requests*: Alice wants to make a request to Bob. In order for her request to be processed by Bob, we require Alice to include several additional information in her request: 1) her access token $access\_token_{Alice}$, 2) receiver's public key $public\_key_{Bob}$, 3) the current time, 4) a set of random bytes - called a nonce - that is supposed to be unique for each request and 5) a signature of the content of the request and the additional information made with $private\_key_{Alice}$. With this information, Bob considers that a request is not legitimate and should not be processed if one of the following cases occurs:
  - Alice's access token $access\_token_{Alice}$ is invalid or expired. To find this out, Bob decrypts $access\_token_{Alice}$ with $public\_key_{auth}$;
  - the signature of the request is invalid after being decrypted with $public\_key_{Alice}$ stored into $access\_token_{Alice}$;
  - the nonce has already been used before;
  - the request's current time field differs from Bob's current time by more than N seconds;
  - the recipient's public key field doesn't match the real $public\_key_{Bob}$.

  These checks protect the exchange against eavesdropped request reuse and man-in-the-middle attacks because Bob is sure that 1) Alice is white-listed and her authorization is still valid, 2) the request was created by Alice and could not have been modified by someone else, 3) Bob is the recipient of the request, and 4) the request is not repeated by someone who eavesdropped a previous request.
- *Responding to requests*: When Bob responds to Alice, we also require Bob to include several additional information in his response: 1) his access token $access\_token_{Bob}$, 2) the nonce sent with Alice's request and 3) a signature of the content of the response and the additional information made with $private\_key_{Bob}$. In the same way as above, a response is not considered valid by Alice if:

- Bob's access token $access\_token_{Bob}$ is invalid or expired after being decrypted with $public\_key_{auth}$;
- the signature of the request is invalid after being decrypted with $public\_key_{Bob}$ stored into $access\_token_{Bob}$;
- the nonce doesn't match the nonce stored into Alice's request;
- the sender's public key field doesn't match the real $public\_key_{Bob}$.

If the response does not check any of the above cases, Alice is sure that 1) Bob is white-listed and still has valid access, 2) the response was sent by Bob and could not be modified, and 3) it is the response to the request associated with this nonce. In short, an eavesdropped response can't be replayed for another request and a man-in-the-middle attacker can't replace the response content.

H.5 Stepwise learning curves

As one can see on Figure 9, collaborative training is nearly equivalent to regular data-parallel training in terms of the total number of SGD updates. The slight difference between the two curves is likely due to random variation, though it can also be explained by the fact that DeDLOC uses *slightly* larger batches due to network latency. In other words, some peers will aggregate a few extra gradients between the moment when the collaboration accumulated 4096 samples and the moment when every peer enters the gradient averaging stage.

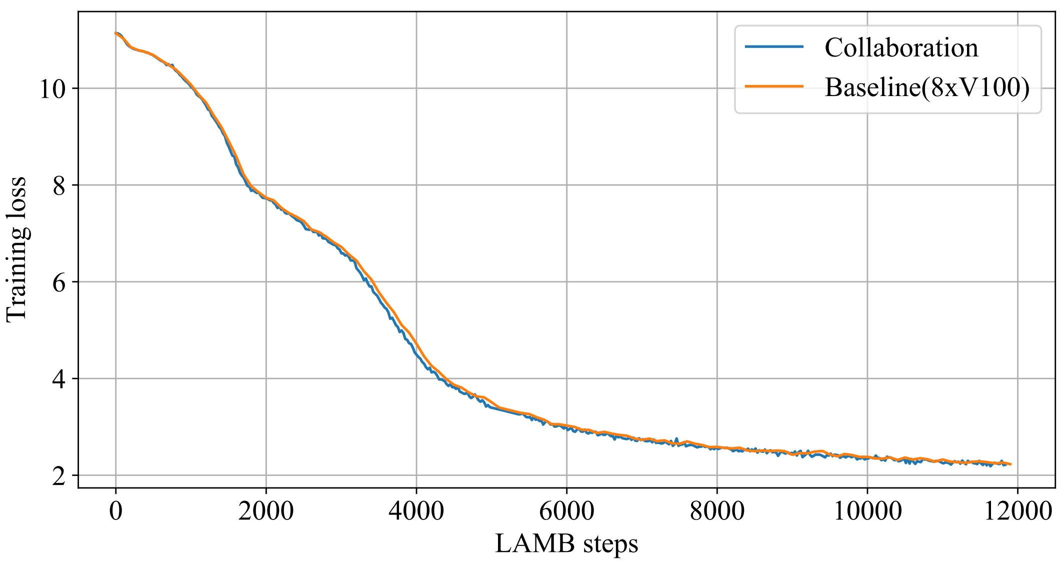


Figure 9: Stepwise learning rate for DeDLOC, compared to regular distributed training.

H.6 Evaluation

We compare sahajBERT with three other pretrained language models: XLM-R [124], IndicBert [108], and bnRoBERTa [109]. For downstream evaluation, we use two tasks from the Indic General Language Understanding Evaluation (IndicGLUE) benchmark [108]: 1) Named Entity Recognition (NER) with the balanced train-dev-test splits version [125] of the original WikiANN dataset [110] and 2) News Category Classification (NCC) with the Soham News Article (SNA) dataset [108].

Each model was finetuned and evaluated as follows:

1. For each tuple of (lr, max_len) in the hyperparameters grid composed of a learning rate lr in (1e-5, 3e-5) and the maximum sequence length max_len in $(64, 128, 192, 256, 512)$, we finetuned the model on the task t and evaluated it on the test set. If t was a NER task, we computed the F1-score, and if it was a NCC task, we computed the accuracy;
2. We repeated the first step three times for different random seeds and computed the mean and standard deviation of the best model metrics in each pool.

All finetuning experiments were ran using the Adam [126] optimizer with the weight decay fix [127], weight decay of 0.001, and a linear decay learning rate schedule. Finally, each model was trained for a maximum number of 20 epochs and stopped earlier if the loss on the validation set did not decrease during 3 epochs. The size of the batch was chosen to be as large as possible: we started with a batch size of 128 and then, if necessary, the batch size is decreased until it can be stored in memory. For exact hyperparameter values, see Table 5.

Table 5: Hyperparameters used for model evaluation.

Task	Model	Learning rate	Input length	Batch size
NER	sahajBERT	1e-05	128	32
	XLM-R	1e-05	256	8
	IndicBERT	3e-05	256	64
	bnRoBERTa	3e-05	512	64
NCC	sahajBERT	3e-05	64	64
	XLM-R	1e-05	128	8
	IndicBERT	3e-05	128	128
	bnRoBERTa	3e-05	128	64

I Environmental impact

Recent works have outlined the environmental consequences of training ever larger deep learning models [128, 129] and encouraged authors to at least report the energy costs incurred [130]. The direction proposed in this work may help in two specific ways. First, while most of the current tools focus on the CO2 cost caused by the training-time energy consumption [111], a more holistic evaluation protocol would need to include the not insignificant manufacturing cost of the training infrastructure [131, 132]. The collaborative training method described here allows volunteers to make better use of existing computing resources, which helps minimize these costs. Second, the distributed training setting allows users to dispense with the extensive cooling infrastructures required for large concentrated data centers, and may thus also help reduce the operating costs themselves [133]. We note however that the additional networking needs may limit the magnitude of these gains.