National Research University Higher School of Economics

as a manuscript

Maksim Riabinin

METHODS AND PROBLEMS OF DECENTRALIZED DEEP LEARNING

PhD Dissertation Summary

for the purpose of obtaining academic degree
Doctor of Philosophy in Computer Science

Moscow – 2023

The PhD dissertation was prepared at National Research University Higher School of Economics.

Academic Supervisor: Artem Babenko, Candidate of Sciences, National Research University Higher School of Economics.

1 Introduction

Topic of the thesis

Over the past decade, deep learning has demonstrated remarkable results, outperforming other machine learning methods on a variety of tasks and domains. Recent years have seen a dramatic growth in the size of neural networks due to a significant impact of the model scale on its resulting capabilities [45; 46]. This presents a challenge to the progress of the broader scientific community: as the resources needed to obtain or exceed state-of-the-art models continue to grow, research in the field becomes less and less accessible to everybody outside of organizations with the most funding. In this work, we argue that a potential solution to this challenge is *decentralization*: instead of obtaining all resources from a centralized high-performance computing (HPC) cluster, we can leverage idle hardware resources of volunteers who are potentially distributed around the globe. Inspired by successes of volunteer computing in other scientific fields [7; 20; 48], we propose deep learning methods that are applicable for general large-scale training and take the unique challenges of volunteer computing into account.

More specifically, this work introduces the *Decentralized Mixture-of-Experts* layer, a sparse neural network architecture that meets the above challenges and naturally handles both node failures and large numbers of irregularly participating peers. Next, we consider training across networks of volunteers in the data-parallel setting: this requires a method that can quickly aggregate model parameters or gradients in presence of network failures. To this end, we develop *Moshpit All-Reduce*, an efficient fault-tolerant method for parameter averaging. Using this method, we propose *Moshpit SGD* — a distributed training algorithm that can be applied to networks of heterogeneous and unreliable devices. Lastly, we propose *Distributed Deep Learning in Open Collaborations*, a practical approach to large-scale collaborative pretraining. This approach combines an adaptive averaging strategy, global gradient accumulation, and careful system design to enable distributed training with workers that have highly diverse network conditions, computational performance, and participation time.

Relevance of the work

The growing size of models is at the heart of many recent advancements in deep learning. Today, the most capable models are routinely reaching the scale of tens and hundreds of billions of parameters [27; 35]: these developments are supported by studies [46] that demonstrate increasing gains in quality or even novel properties [18] of neural networks at larger sizes. Correspondingly, the size of training datasets is also growing: as recent works suggest [54], the number of examples might be equally as important as the model size when training a neural net-

work with a fixed compute budget. Both of these scaling directions require an immense amount of computational resources: all state-of-the-art models are trained in HPC clusters with hundreds or even thousands of specialized accelerators and dedicated high-speed networking solutions.

Predictably, acquiring the computational resources to train such large models can be difficult for an average researcher. Renting even one deep learning accelerator for a month may cost several thousands of dollars, and building a cluster is often outside the budget constraints for organizations with modest funding. This dramatically limits the availability of state-of-the-art research to a set of laboratories that can afford to run large-scale experiments with billion-scale neural networks. In turn, this results in a smaller potential for replicating or adapting the latest results to new datasets, an inability to analyze or improve the training process of large models, and overall difficulties in contributing to further scientific progress in deep learning.

In this work, we explore an alternative approach to large-scale deep learning that does not involve expensive supercomputers. We take inspiration from successful cases of leveraging volunteer resources in other sciences, such as computational biology [20] or astrophysics [17]. The most famous example of such projects is Berkeley Open Infrastructure for Network Computing (BOINC) [7], which became the first "supercomputer" reaching the exaflop scale [19]. However, directly applying existing methods for distributed deep learning in such conditions is challenging because of multiple infrastructure-related challenges.

Specifically, the most popular methods for efficient distributed training [22; 40; 47] are not designed to handle node failures or connectivity issues: in the most severe cases, even one disconnected peer can jeopardize the entire training procedure or significantly inhibit its progress. At the same time, workers in a volunteer computing setup possess a much higher degree of heterogeneity: each personal computer might have a unique hardware and networking setup, and this diversity needs to be taken into account when designing such decentralized training systems. Lastly, the communication links across cluster nodes can be magnitudes faster than standard Internet connections of collaborative experiment participants, which also impacts our design choices. Hence, we develop methods that aim to maximize the distributed training performance in the conditions outlined above.

The first work described in this thesis focuses on the goal of training models that can exceed the limits of a single device in the context of decentralized training. Trading off generality for performance, this work introduces *Decentralized Mixture-of-Experts* (DMoE), which is a specialized layer designed to be sharded across the computers of volunteers. Similarly to standard Mixture-of-Experts models [2], the DMoE layer consists of independent sublayers called *experts* that get assigned to the input based on the output of the *gating function*. We propose a natural extension of this architecture for fault-tolerant training and show that DMoE is not sensitive to communication latency. Another important difference is that the DMoE experts are located by

other nodes using distributed hash tables (DHT), a fault-tolerant decentralized key-value storage. This mitigates the need for a centralized entity that would track available experts, which might not be feasible in larger collaborations without incurring significant costs. To efficiently find the most relevant experts for a given input, we propose a *structured gating function* that factorizes the set of experts in a predefined multidimensional grid.

The subsequent part of this work addresses the problem of data-parallel training with volunteers. Our rationale for that is twofold: first, even if we use mixture-of-experts in each model layer, we still need to have parameters of the gating function and the embedding layer that are consistent across the collaboration. Second, with memory-efficient training methods (such as lower numeric precision [32] or parameter sharing [3]), it might be possible to train models that can fit consumer GPUs yet still require large amounts of computation to achieve the best quality.

In the second paper covered in this thesis, we study methods for efficiently aggregating the model gradients for distributed training. The family of communication-optimal methods, known as All-Reduce [37], is not fault-tolerant by default and thus unsuitable for our goals. On the other hand, more robust methods for decentralized training, such as Gossip [49; 52], require many communication rounds to achieve consistency across the network. We propose *Moshpit All-Reduce*, an iterative averaging algorithm that combines the fault tolerance of Gossip-based methods with the efficiency of All-Reduce. It combines the participants into independent groups and ensures that peers within one group are assigned to different groups in the next round. *Moshpit SGD*, a distributed optimization algorithm based on Moshpit All-Reduce, has convergence rates equivalent to standard distributed SGD (more specifically, Local-SGD [51]) yet exhibits much higher large-scale training performance in slower networks with node failures, as we demonstrate in our experiments.

Finally, the third work presents *Distributed Deep Learning in Open Collaborations* (DeDLOC), an approach that takes node heterogeneity into account and alleviates the issue of slower communication speeds of volunteer-oriented distributed training. Specifically, we propose an adaptive averaging strategy that assigns training and gradient aggregation tasks to workers based on their performance to minimize the overall time of averaging, the fundamental communication phase in data-parallel training. We also design a decentralized tracking mechanism for the total accumulated batch size, which is necessary to enable the dynamic participation of peers. Aside from ablation studies, the paper presents the results of the first collaborative language model pretraining experiment: an effort organized by the authors and a community of volunteers has resulted in *sahajBERT*, a Bengali masked language model that has competitive performance with both monolingual and multilingual baselines [25; 56].

Moreover, the methods we develop can be applied not only in the volunteer computing scenario. Specifically, cloud providers frequently offer *preemptible* (or *spot*) instances at a cost

that can be 2–3 times lower than the cost of on-demand servers [5; 21]. Spot instances, however, have the disadvantage of non-guaranteed availability: if the demand for nodes with their hardware configuration increases, some of these instances might become unavailable until the demand recedes. In principle, these conditions make applying traditional high-performance distributed methods infeasible. Usually, efficient training relies on reliable uptime and high communication speeds, both of which are difficult to achieve in preemptible environments. Still, the target setting of this work considers most challenges that arise from using spot instances. Hence, as we show in our experiments below, the proposed methods can be applied to heterogeneous volunteer hardware and to more homogeneous, yet still unstable, preemptible cloud servers.

The goal of this work is to develop practical large-scale distributed training methods for slowly-connected networks consisting of heterogeneous and unreliable nodes.

2 Key results and conclusions

The contributions of this work can be summarized as follows:

1. We proposed *Decentralized Mixture-of-Experts* (DMoE), a neural network layer designed for training large models in volunteer computing conditions. To handle peer failures and low-speed communication in a unified framework, we designed *Learning@home* — a system for large-scale learning in the setting of volunteer computing. We empirically validate both the performance of Learning@home under latency and the convergence of DMoE models in presence of node failures to the same results as equivalent dense models.

2. We proposed *Moshpit All-Reduce*, an efficient decentralized gradient averaging method, and *Moshpit SGD*, a distributed optimization algorithm that leverages Moshpit All-Reduce, has equivalent convergence rates to Local-SGD, and is suitable for training with unreliable devices. In large-scale training experiments, Moshpit SGD outperforms existing baselines (including previous decentralized methods) tasks by more than 30% in terms of wall clock time until convergence to the target loss value.

3. We proposed *Distributed Deep Learning in Open Collaborations* — a practical method for collaborative data-parallel deep learning. This method is robust to peers joining and leaving during training and takes the diversity of hardware setups and network connections into account by design. We also ran a real-world collaborative experiment and train *sahajBERT*, a neural network for Bengali language representations with downstream results that are comparable to regular models trained in clusters which cost ≈ 3 times more.

Theoretical and practical significance.

We propose several methods that can be applied for large-scale training of neural networks in decentralized setups with unstable connections between heterogeneous and intermittently available nodes. Such conditions can arise in two cases: the first one is collaborative training (among organizations or simply volunteering individuals), and the second one is cost-efficient training with preemptible instances. The research described in this work aims to make these two cost-efficient alternatives to HPC more practical and widespread. All of our implemented methods have publicly available open source implementations in PyTorch [41], one of the most popular deep learning frameworks at the time of writing. This makes it possible for any deep learning practitioner to apply these methods and orchestrate decentralized training experiments.

Key aspects/ideas to be defended.

1. The *Distributed Mixture-of-Experts* layer for training large neural networks using volunteer resources.

2. The *Moshpit All-Reduce* protocol and the *Moshpit SGD* algorithm for communication-efficient training across unstable devices.

3. *DeDLOC*, an approach for data-parallel pretraining in large collaborations consisting of nodes with highly diverse capabilities.

Personal contribution.

In "Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts", the author of this work designed the core idea behind the approach, implemented the runtime component of Learning@home, ran all experiments and wrote most of the paper (aside from discussions on the Distributed Hash Table).

In "Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices", the author proposed Moshpit All-Reduce, ran the majority of averaging experiments in Section 4.1 and all pretraining experiments, obtained Theorem C.1, and proposed the load balancing approach in Appendix G (which was later developed into a more general version in DeDLOC by Michael Diskin).

In "Distributed Deep Learning in Open Collaborations", the author designed the core idea of research and led the project, implemented the original codebase for pretraining ALBERT models, conducted the sahajBERT collaborative experiment, and wrote the majority of the paper.

Publications and approbation of the work

* denotes equal contribution

First-tier publications

1. **Max Ryabinin***, Anton Gusev. Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts. In Advances in Neural Information Processing Systems, 2020 (NeurIPS 2020). Pages 3659–3672. CORE A* conference.

2. **Max Ryabinin***, Eduard Gorbunov*, Vsevolod Plokhotnyuk, Gennady Pekhimenko. Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices. In Advances in Neural Information Processing Systems, 2021 (NeurIPS 2021). Pages 18195–18211. CORE A* conference.

3. Michael Diskin*, Alexey Bukhtiyarov*, **Max Ryabinin***, Lucile Saulnier, Quentin Lhoest, Anton Sinitsin, Dmitry Popov, Dmitry Pyrkin, Maxim Kashirin, Alexander Borzunov, Albert Villanova del Moral, Denis Mazur, Ilia Kobelev, Yacine Jernite, Thomas Wolf, Gennady Pekhimenko. Distributed Deep Learning In Open Collaborations. In Advances in Neural Information Processing Systems, 2021 (NeurIPS 2021). Pages 7879–7897. CORE A* conference.

Reports at conferences and seminars

1. Invited talk on "Learning@home: Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts". Eindhoven Reinforcement Learning Seminar, virtual, April 30 2020.

2. Poster presentation on "Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts". Neural Information Processing Systems, virtual, December 6 2020.

3. Report on Decentralized Deep Learning. Seminar of the Faculty of Computer Science of HSE University in Voronovo, May 29 2021.

4. Invited talk on "Decentralized Deep Learning: Training Large Neural Networks Together". Second Workshop on Distributed Machine Learning (DistributedML), virtual, December 7 2021.

5. Poster presentation on "Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices". Neural Information Processing Systems, virtual, December 10 2021.

6. Poster presentation on “Distributed Deep Learning In Open Collaborations”. Neural Information Processing Systems, virtual, December 10 2021.

7. Invited talk on “Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices” at “Endless Summer School: NeurIPS Highlights”. Vector Institute, virtual, February 16 2022.

8. Invited talk on “Decentralized Deep Learning: Training and Running Large Models over the Internet”. DeepMind, virtual, December 12 2022.

9. Invited talk on “Decentralized Deep Learning: Training and Running Large Models over the Internet”. ETH Zurich, December 20 2022.

10. Invited talk on “Decentralized Deep Learning: Training and Running Large Models over the Internet”. Naver Labs Europe, February 7 2023.

11. Invited talk on “Decentralized Deep Learning: Training and Running Large Models over the Internet”. Institute of Science and Technology Austria, March 21 2023.

Volume and structure of the work. The thesis contains an introduction, contents of publications and a conclusion. The full volume of the thesis is 132 pages.

The author has also contributed to the following publications:

1. Alexander Borzunov*, **Max Ryabinin***, Tim Dettmers*, Quentin Lhoest*, Lucile Saulnier*, Michael Diskin, Yacine Jernite, Thomas Wolf. Training Transformers Together. In Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track. Pages 335–342.

2. Eduard Gorbunov*, Alexander Borzunov*, Michael Diskin, **Max Ryabinin**. Secure Distributed Training at Scale. In Proceedings of the 39th International Conference on Machine Learning 2022 (ICML 2022). Pages 7679–7739.

3 Content of the work

3.1 Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts

Decentralized Mixture-of-Experts

To train large models in the setting of volunteer computing, we propose Decentralized Mixture-of-Experts (DMoE), a direct extension of standard mixture-of-experts (MoE) layers [2; 34] for the setting of decentralized deep learning. Each DMoE layer contains multiple *experts* — independent parallel units that have the same architecture but different weights. On the forward pass, each input example is routed to most relevant experts by means of a *gating function*, which is a trainable classifier that determines the priority of an expert for an input. Similarly to regular MoE, DMoE can process any kind of input data by using appropriate types of layers as experts. The key difference of Decentralized Mixture-of-Experts is that experts are distributed over the network of volunteer computers according to the memory limits of each device.

To share necessary metadata across the network, we use Distributed Hash Tables (DHT, [11]), a distributed key-value data structure that requires no centralized coordination, is designed to be fault-tolerant and has logarithmic scaling with respect to the number of nodes. These properties have made DHT a popular choice for robust data storage in peer-to-peer applications; we use Kademlia [31] as one of the most popular DHT protocols with existing public implementations. For DMoE, DHT stores the expert metadata, such as the corresponding worker status and its location. An example of DMoE forward and backward pass is given in Figure 1.

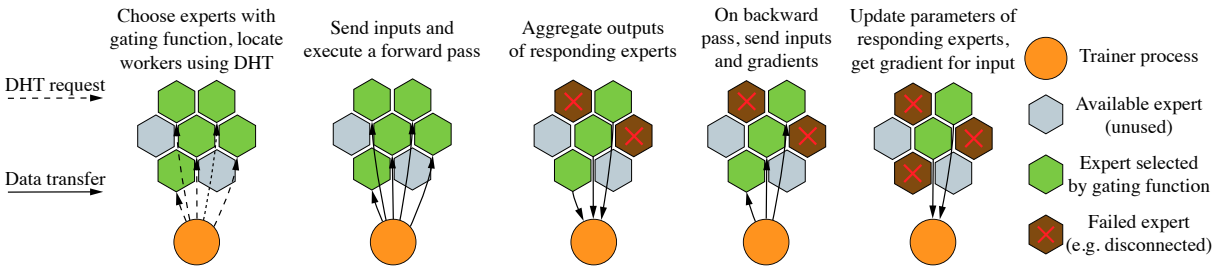


Figure 1: DMoE layer processing a single example with peer failures.

Importantly, using DMoE layers can resolve multiple issues that arise with volunteer computing. First, if the requested expert fails to respond, DMoE will be able to exclude it from averaging by renormalizing the weights of other experts. Second, servers with different memory capacities can accommodate different numbers of experts. Next, increasing the diversity of mixture-of-experts layers to help expert specialization [34] serves as a natural way of balancing the load between different servers. Finally, the sparse nature of mixture-of-experts models

makes them less sensitive to stale gradients that occur in asynchronous training, which is highly useful for increasing the training throughput in case of high network latency. Strictly speaking, if the weights of the neural network were changed by the optimization step, its gradients for previous iterations computed on other nodes become invalid. Thus, having independently updated sets of weights for each expert can reduce the degree of overlap for concurrent updates.

To find the highest-scoring experts in an efficient manner over a large network of volunteers, we propose a structured gating function, which has a design similar to product key layers [29]. This function operates over a set of experts arranged into a grid of $d \times M$ elements: as we might expect new volunteers to join over time, it should be sufficiently large and have empty positions to allocate future experts. Importantly, this grid structure associates each expert f with its unique identifier, allowing to encode M^d experts in Md space:

$$\text{uid}(f) = (u_0, u_1, \ldots, u_{d-1}), u_i \in [0, M). \tag{1}$$

Hence, by arranging experts in this way, we only need to predict Md values for expert priorities. Specifically, the gating function $g(x, f)$ consists of d linear layers with M outputs, and the priority of each expert is computed as a sum of grid values corresponding to its identifier:

$$g(x, f) = \sum_{i=1}^{d} g_i(x)_{u_i},\ u_i \in \text{uid}(f). \tag{2}$$

As a result, we can select approximate top-k highest responding experts in $O(dk \log N)$ time (N is the total number of peers) using the beam search algorithm. First, we predict the priorities for all grid dimensions, and then traverse the grid by increasing i from 1 to d. We maintain the list of k highest-scoring identifier prefixes for each iteration, which gives us the answer after the end of the traversal. For each dimension, we first expand the candidate list by finding all experts with given prefixes that exist in the DHT, and then save k highest-scoring continuations as the candidate list.

When the most relevant experts are found, we find their respective servers using the DHT and send them the vector of input activations. After the requested experts have responded with their outputs, we average their responses with weights determined by the gating function:

$$\text{DMoE}(x) = \sum_{f \in \text{TopK}(x)} f(x) \frac{\exp\left(g(x, f)\right)}{\sum_{f' \in \text{TopK}(x)} \exp\left(g(x, f')\right)}\text{, TopK}(x) \text{ are } k \text{ best experts w.r.t. } g \tag{3}$$

Learning@home

To support training of large models on volunteer hardware, we also develop Learning@home, an infrastructure that allows efficient training in the presence of node failures and slow network speeds. It consists of three main components:

- **Trainer** generates the batches of data and performs forward and backward passes through the entire model.
- **Runtime** serves experts and processes incoming requests for forward and backward passes (including the optimization step in case of the backward pass). To maximize its throughput, the runtime aggregates incoming requests of the same type into batches.
- **DHT Node** announces experts and exchanges their metadata with other nodes.

Empirical results

First, we evaluate the throughput (number of examples processed per second) of Learning@home in different network conditions. We emulate the distributed training environment by serving large feed-forward layers or Transformer [10] encoder blocks over 4 GPUs, simulating network delays of varying degrees. Our baseline for performance here is non-asynchronous model parallel training [22], and we include the result of training without network delays as our upper bound on performance. As we can see in Figure 2, the throughput of our approach remains consistent even in high-latency environments, which validates its utility in our setting. We also conduct a comparison in a more realistic setup of three cloud GPU instances in different regions: Table 1 depicts results that are similar to simulated experiments, confirming our findings.

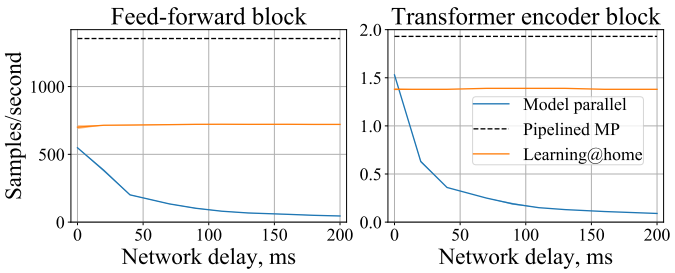


Figure 2: Training throughput as a function of latency.

Approach	Feed-forward	Transformer
Model-parallel	7.23 ± 0.06	0.01 ± 0.0
Learning@home	300.8 ± 15.9	0.68 ± 0.0

Table 1: Training throughput (measured in examples/s) for 3 servers in different regions.

Our second set of experiments verifies the robustness of DMoE to asynchronous updates and gradient staleness. We compare DMoE with a sequence of dense layers having the same forward pass computational cost as a subset of 4 experts. To evaluate different network conditions, we consider three setups: low latency (100ms delay for each request), high latency (1000ms delay), and high latency with failures (0.1 probability of not responding to a request).

We train models on two datasets: a sequence of feed-forward layers on MNIST [23], and Transformer-XL [55] on WikiText-2 [50]. In the case of WikiText-2, we only consider the setting of high latency with failures. The results are shown in Figure 3 and Figure 4: as we can see from the plots, the gradient staleness introduced by asynchronous training in high latency setups affects DMoE to a lesser degree compared to regular dense models.

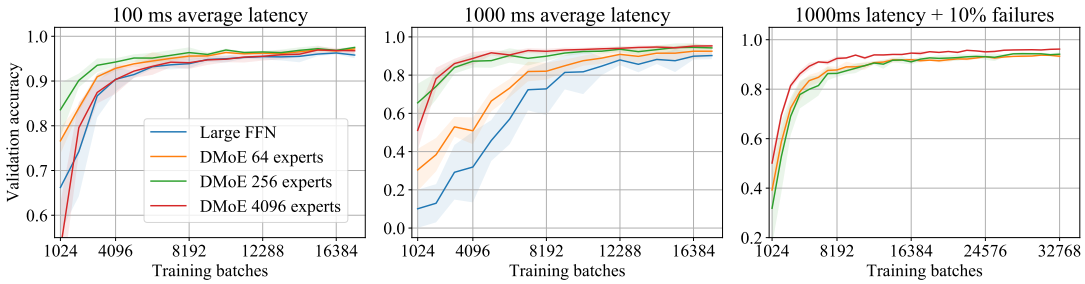


Figure 3: Convergence plots for feedforward models trained on MNIST with different network latencies and failure rates. Light areas depict standard deviations over 5 runs.

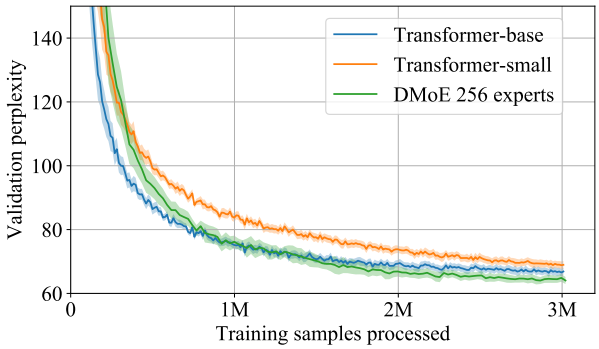


Figure 4: Convergence plots for Transformer-XL language models trained on the WikiText-2 dataset. Light areas depict standard deviations over 5 runs.

3.2 Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices

To run distributed training in a *data-parallel* setting (with all peers processing different data), it is necessary to have an algorithm for aggregating updates across the network. For example, in case of Decentralized Mixture-of-Experts, the input embeddings and the gating function parameters are still shared between workers. In its simplest form, training a neural network reduces to a stochastic optimization problem of minimizing the expected loss function $f(x, \theta)$ depending on inputs x and model parameters θ with respect to θ:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} f(x_i, \theta) \tag{4}$$

As a result, the primary communication pattern of data-parallel deep learning is averaging the gradients $\nabla_\theta f(x_i, \theta)$ across the network nodes. Hence, training with unreliable participants requires a communication-efficient algorithm that is robust to node failures. However, most widely used methods have only one of these properties: All-Reduce is theoretically optimal in terms of network transfer [37] but not fault-tolerant, and Gossip-based approaches [13; 43; 52] to decentralized training exchange parameters over a sparse graph, avoiding the need for a

simultaneous synchronization across the entire network but communicating less efficiently. We propose Moshpit All-Reduce, an averaging method for decentralized deep learning that is highly efficient yet fault-tolerant.

Moshpit All-Reduce

Running Moshpit All-Reduce involves executing several iterations of averaging in smaller dynamically changing groups that do not overlap within each iteration. Intuitively, in this case, the failure of a single peer leads to an interruption only within a small subgroup of the network. Each group consists of peers sharing the same coordinate in a virtual d-dimensional grid with M elements in each dimension. In practice, M and d are chosen to accommodate all participating peers with a relatively high grid utilization.

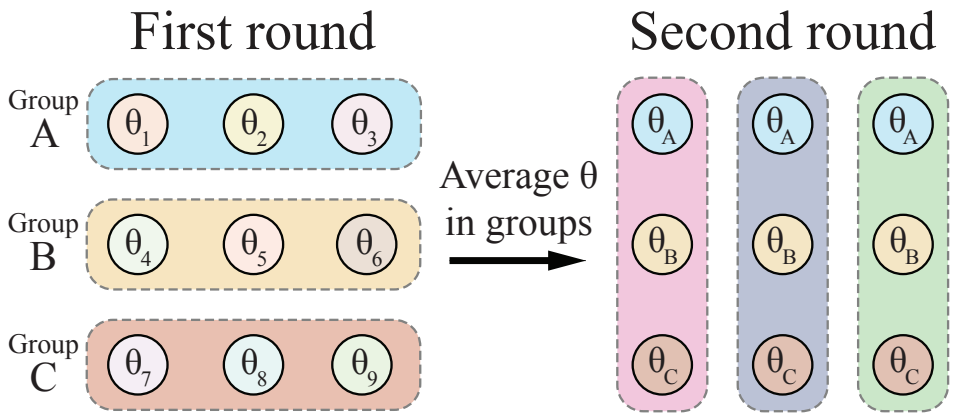


Figure 5: Example steps of Moshpit All-Reduce for 9 peers executed in 2 iterations.

The method is illustrated in Figure 5 and described formally in Algorithm 1. Peers find the next group for averaging using *grid indices* C, which are formed as multidimensional integer tuples. These tuples are initialized using the `get_initial_index` function that samples indices for each dimension from a uniform distribution. On each iteration of the procedure, nodes communicate using the DHT to announce their network addresses and current grid indices. Then, they organize into groups and get the list of immediate peers by also using the DHT. In the next step, peers within the same group exchange data using Butterfly All-Reduce [42]: this procedure, shown in Figure 6, assigns each peer to aggregate a separate part of the averaged vector. Finally, peers update their group indices based on the indices of Butterfly All-Reduce parts that they were responsible for: this ensures a different set of neighbors for the next step, as no peers share a part index within one All-Reduce iteration.

Algorithm 1 Moshpit All-Reduce for peer i

Input: parameters $\{\theta_j\}_{j=1}^N$, number of peers N, d, M, number of iterations T, peer index i
$\theta_i^0 := \theta_i$
$C_i^0 := \texttt{get\_initial\_index(i)}$
for $t \in 1 \dots T$ **do**
  $\texttt{DHT}[C_i^{t-1}, t].\texttt{add}(\texttt{address}_i)$
  $\texttt{Matchmaking()}$
  $\texttt{peers}_t := \texttt{DHT.get}([C_i^{t-1}, t])$
  $\theta_i^t, c_i^t := \texttt{AllReduce}(\theta_i^{t-1}, \texttt{peers}_t)$
  $C_i^t := (C_i^{t-1}\texttt{[1:]}, c_i^t)$
end for
Return θ_i^T

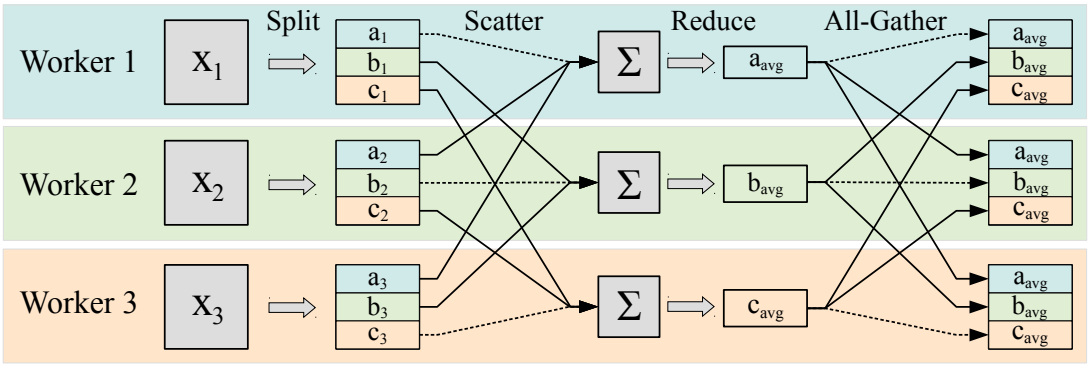


Figure 6: The Butterfly All-Reduce algorithm. First, peers split the averaged vector into equal segments. Then, each peer aggregates a particular segment across all nodes. Finally, peers send the averaged parts to all other nodes to get the result.

If the grid is fully occupied ($N \equiv M^d$) and all All-Reduce calls are successful, Algorithm 1 becomes a generalization of Torus All-Reduce [44] and computes the exact average after d iterations, as we prove in the theorem below:

Theorem 1. *Assume that M^d peers are arranged in a d-dimensional hypercube with M positions in each dimension. Also, assume that each peer fully participates in every averaging step, and M-sized groups for each averaging iteration are determined based on the hypercube coordinates. Then, if Moshpit All-Reduce is ran in the above setup for d iterations without repeating groups (i.e. averaging across each dimension exactly once), its result for each participant is the average value of θ across all M^d peers.*

Still, these assumptions are unlikely to hold in practice for the entire duration of training. More specifically, volunteers will join and leave during the experiment, possibly during the All-

Reduce phase; also, it can be challenging to ensure that the total number of participants can be arranged in a grid of the required structure without any gaps. However, as we show in the paper, Moshpit All-Reduce can asymptotically reduce the inconsistency between peers even in the setting of non-constant participation and with uneven groups.

Finally, if peers in the same group have uneven network bandwidth, assigning equal parts of the aggregated vector in Butterfly All-Reduce will cause communication bottlenecks and unnecessary delays in the training procedure. To prevent this, we can dynamically adjust the communication load of each peer. Specifically, we design an optimization problem that allocates differently sized fractions w_i of the averaged vector to M peers with different network bandwidth b_i. We minimize the total averaging time, which can be modeled as the maximum time across all nodes to receive, aggregate, and send the corresponding vector part and send (and receive) the rest of the vector for aggregation by other peers.

The resulting optimization problem is given below:

$$\begin{aligned} \min_{w} \quad & \max_{i} \left(1 - w_i + (M-1)w_i\right) \cdot \tfrac{1}{b_i} \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{M} w_i = 1, \\ & w_i \geq 0 \qquad\qquad \forall i \end{aligned} \tag{5}$$

As we show in the paper, it is possible to reduce this to an instance of a linear programming problem, which allows us to use efficient solvers [6] and find the optimal averaging strategy in a negligible amount of time.

Moshpit SGD

To run distributed training on networks composed of unreliable participants, we formulate Moshpit SGD, a modification of standard stochastic gradient descent with local steps [36]. During the communication phase, peers leverage Moshpit All-Reduce for parameter averaging instead of using standard All-Reduce or a parameter server [30]. Importantly, under mild assumptions, we can derive convergence rates for Moshpit SGD that are competitive with state-of-the-art results [1; 14] at the time of publication.

Empirical results

First, we verify the efficiency and robustness of Moshpit All-Reduce that we aimed to obtain when developing this method. To do this, we run simulated averaging experiments, using a sample from the standard normal distribution as input data for each worker. We consider several settings: 1024 peers with no failures, 1024 peers with a failure rate of 0.005, and 768 peers with a failure rate of 0.005 (used to show the impact of having a non-full grid). We report the

mean squared deviation of data on all workers (or, equivalently, the variance between nodes): a successful exact averaging procedure results in an error of zero.

We use several algorithms from prior work as baselines: the list includes All-Reduce (with restarts after failures), Gossip [13], and PushSum [53]. In addition, we report the performance of a simplification of Moshpit All-Reduce that averages parameters in small *random* groups.

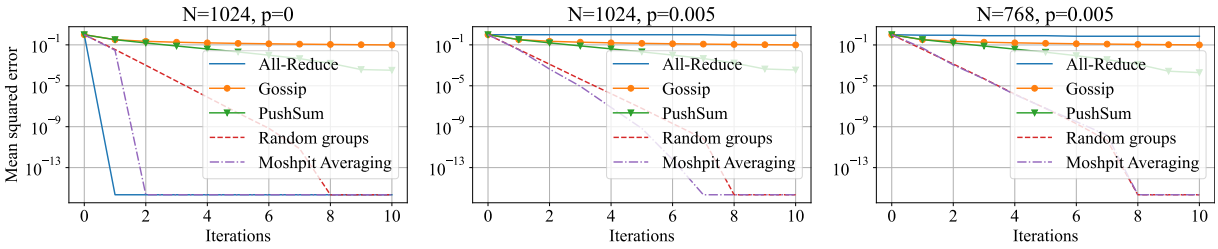


Figure 7: Mean squared error of averaging methods as a function of the iteration number.

The main results of these experiments are shown in Figure 7. As expected, standard All-Reduce computes the average in a single step when there are no peer failures. However, it fails to average the values on workers in less optimistic cases; Gossip and PushSum also require a significant number of iterations for convergence. On the other hand, Moshpit SGD both computes the exact average in 2 iterations in case of no failures and converges faster than other methods in case of a non-zero failure probability. Random group averaging has performance similar to Moshpit All-Reduce for low grid utilization but underperforms our method for fuller grids.

Also, we benchmark Moshpit SGD in several distributed training experiments, comparing its performance with several popular algorithms for large-scale deep learning. We consider two settings covering different application domains: ResNet-50 [16] image classifier training on ImageNet [24] and ALBERT-large [3] language representation model pretraining on BookCorpus [4]. Along with Moshpit SGD, we evaluate multiple baselines, including standard All-Reduce SGD, as well as two decentralized methods: AD-PSGD [9] and Stochastic Gradient Push [52]. Our training setups encompass both the case of homogeneous training (a single multi-GPU server) and heterogeneous training across different environments and locations.

The outcome of these experiments can be seen in Figure 8: as we demonstrate, Moshpit SGD outperforms all baselines in terms of time until convergence, including the best decentralized ones. Importantly, this *does not mean* that the iteration convergence time is the smallest as well: All-Reduce SGD (AR-SGD) converges in fewer epochs due to more accurate gradient estimates at each step. For training ALBERT, the heterogeneous setup uses less powerful preemptible instances and is thus more cost-efficient than the homogeneous setup: however, the instability of nodes makes it impossible to train with All-Reduce in this setting. Conversely, the fault tolerance of Moshpit SGD makes it possible to achieve the same training loss in a smaller amount of time.

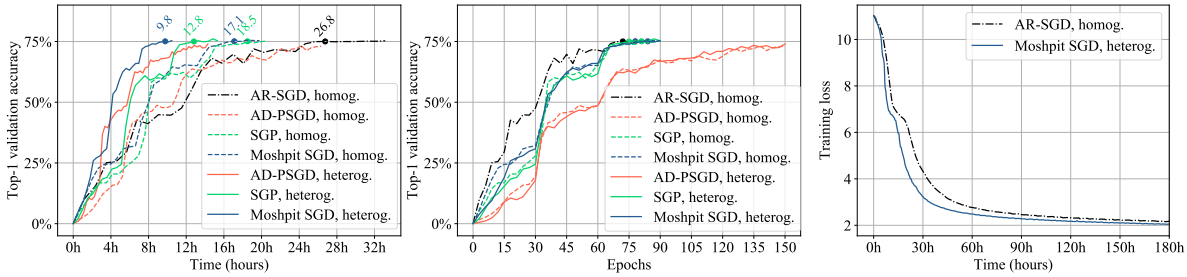


Figure 8: **(Left, Middle)** ResNet-50 accuracy on ImageNet validation set as a function of wall clock time (left) and training epochs (middle). **(Right)** Training loss convergence of ALBERT-large.

3.3 Distributed Deep Learning in Open Collaborations

Although Moshpit SGD can be used even on its own for training with volunteers, handling large numbers of participants in an efficient, inclusive, and robust manner requires resolving several additional problems. For example, in a collaborative setting, one might expect a dynamic composition of participants and hence a dynamically changing set of computational and communication capabilities for each node in the network. In addition, the dynamic participation of peers might make it non-trivial to ensure that the training results are comparable (ideally, equivalent) to the outcomes obtained in a regular HPC cluster. To address these considerations, we design Distributed Deep Learning in Open Collaborations (DeDLOC), a comprehensive decentralized approach for large-scale data-parallel training with volunteers.

Method description

The relatively easier challenge is that of achieving consistent training results. As most state-of-the-art models are usually trained with large batches [27; 28; 39; 46; 58], we leverage the same approach, accumulating gradients for processed examples across the entire collaboration before making a synchronous step on all nodes. Importantly, this offers a natural way of supporting dynamic participation in real-world conditions: if a peer leaves the experiment during training, other peers will compensate for that by processing more examples, which will delay but not prevent the next SGD step. Conversely, if a new peer joins the experiment, the computational throughput of the collaboration grows, and the time between subsequent steps decreases. An illustration of this inherent elasticity of large-batch training can be seen in Figure 9.

Notably, we assume that all peers process data coming from the same distribution: otherwise, the online peers would not be able to compensate for the disconnected ones. Still, for large-scale pretraining, the data is usually obtained from public sources and can exceed hundreds of gigabytes in size, so this assumption is quite realistic for the setting that we consider.

This approach allows us to have the same training updates and thus, the same learning dynamics as in a standard data-parallel setting from an algorithmic viewpoint. Hence, if the training step and the gradient aggregation procedure are the same as in the regular centralized setting, DeDLOC will reach results that are similar to the results of standard methods up to numerical precision (and other sources of instability frequent in deep learning). In practice, we still relax this constraint to some degree: we can use group averaging techniques like [12] or Moshpit All-Reduce for improved fault tolerance and Delayed Parameter Updates (DPU [59], also known as one-step delayed SGD [8]) to overlap a costly communication phase with computing the gradients for the next step. Importantly, both the analysis and the empirical evidence demonstrate that these modifications have little impact on training convergence.

Next, we develop an adaptive algorithm for data-parallel training that can dynamically adjust the aggregation strategy given the current network and hardware conditions of the collaboration. This is necessary to achieve optimal performance in a heterogeneous setting with dynamic peer participation: having a uniform distributed strategy is suboptimal for peers with highly different performance. Moreover, nodes with a fast Internet connection might have a relatively slow compute accelerator and vice versa.

For this purpose, we design an optimization problem that maximizes the training throughput. In the case of DPU, the throughput can be viewed as the minimum of the batch processing time for B examples and the gradient aggregation time for a neural network with P parameters. Our primary constraints are the computational performance of each peer i (gradients for s_i examples obtained per second), the download and upload speed of peers (d_i and u_i correspondingly), and the pairwise bidirectional transfer limitations (t_{ij} and t_{ji} for peers i and j). We optimize the averaging strategy with respect to the peers computing gradients (indicator variables c_i), the speed of sending local gradients for aggregation a_{ij}, and the speed of sending aggregated gradients back to peers g_{ij}.

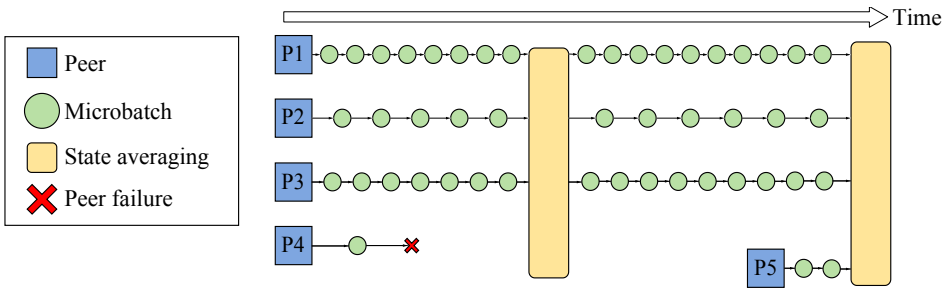


Figure 9: Two example training steps of DeDLOC with peers that are joining and leaving. First, peer 4 leaves, so others needs to process more examples to reach the same batch size. In the next iteration, peer 5 joins and the time before the target batch size is accumulated becomes smaller.

The final optimization problem is given below:

$$
\begin{aligned}
\max_{a,g,c} \quad & \min\left(\frac{\sum_{i=1}^{n} s_i \cdot c_i}{B},\ \frac{\min_i \sum_j g_{ji}}{P}\right) & \\
\text{subject to} \quad & g_{ij} \le \min_{k \in \{r:c_r=1\}} a_{ki} & \forall i,j \\
& \sum_{j \neq i} (a_{ji} + g_{ji}) \le d_i & \forall i \\
& \sum_{j \neq i} (a_{ij} + g_{ij}) \le u_i & \forall i \\
& a_{ij} + g_{ij} \le t_{ij} & \forall i,j \\
& a_{ij},\ g_{ij} \ge 0, c_i \in \{0,1\} & \forall i,j
\end{aligned}
\tag{6}
$$

An example admissible solution for this problem is $c_i = 1$ only for the peer with highest s_i and $a, g \equiv 0$, which corresponds to no communication between peers. However, to find the most efficient training strategy for a given composition of peers, we need to find an optimal solution for each training iteration. Thus, we reformulate the optimization problem as a linear program, which allows us to find a solution in sub-second time using existing solvers [6].

Notably, in special cases (such as homogeneous network and hardware conditions), the optimal solution corresponds to well-known distributed training paradigms, as our strategy is general enough to accommodate them. Figure 10 displays an illustration of several strategies that are optimal in different conditions.

Empirical results

In our first group of experiments, we validate the adaptive properties of DeDLOC by running it in a series of setups with varying heterogeneity. We focus on self-supervised machine learning models: they both require a substantial amount of resources for training and can be applied to multiple tasks, potentially making them highly attractive for volunteer deep learning.

We begin with measuring the performance of training Swapping Assignments between multiple Views (SwAV), a method for learning universal image representations [57], on ImageNet. We consider three setups: *Server* with 8 nodes having one V100 GPU each and 1 Gb/s network, *Workstation* with 16 workers having one 1080 Ti GPU and 200 Mb/s network, and *Hybrid* that combines them. For averaging benchmarks, we also include the fourth setup that adds a single

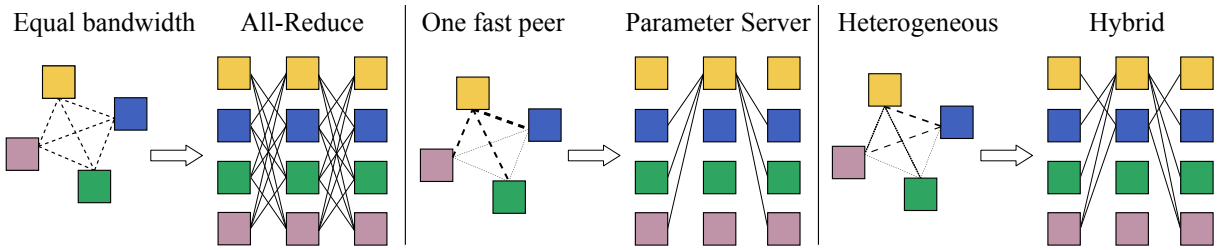


Figure 10: Example hardware and network conditions for participating peers along with optimal averaging strategies. Bolder lines correspond to faster connections.

high-bandwidth node. Our key performance metrics are the training performance as a function of time, as well as the gradient averaging speed. The results of these experiments are available in Figure 11 and Table 2. Most importantly, our hybrid averaging strategy yields speedups of up to 92% in heterogeneous networks compared to the All-Reduce and Parameter Server baselines while being close to the optimal method in all tested setups.

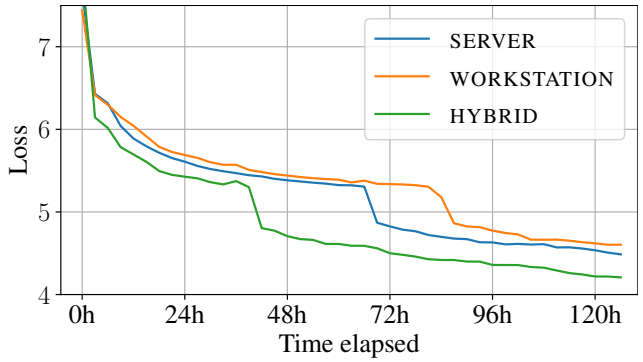


Figure 11: SwAV convergence in different setups.

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

Table 2: SWaV averaging performance.

Next, we evaluate the performance of pretraining ALBERT-large [3] on WikiText-103 [38]. We compare the training loss convergence in five setups: *High-bandwidth* has 16 workers with a T4 GPU and 25 Gb/s bandwidth, the *Heterogeneous* one has the same GPUs with 4x 200 Mb/s, 8x 100 Mb/s, and 4x 50 Mb/s networks; in *Heterogeneous + load balancing*, we add load balancing from our adaptive averaging algorithm, and *CPU-only* and *Time-varying* setups add nodes with a fast network but no GPUs and nodes with unstable participation, respectively.

The results of this evaluation are shown in Figure 12: most importantly, having a naïve averaging algorithm in heterogeneous conditions significantly slows down training, but an adaptive strategy brings the performance closer to that of a regular HPC setup. Having additional peers that assist with averaging or participate part-time helps reduce the training time even further.

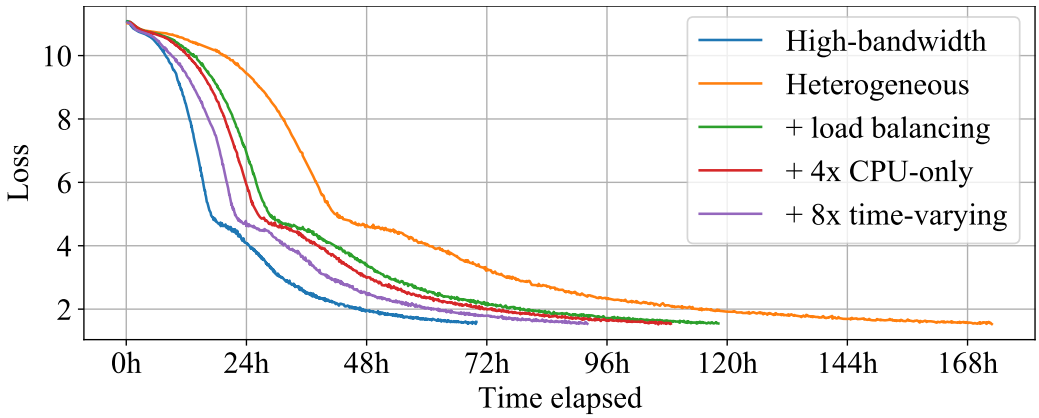
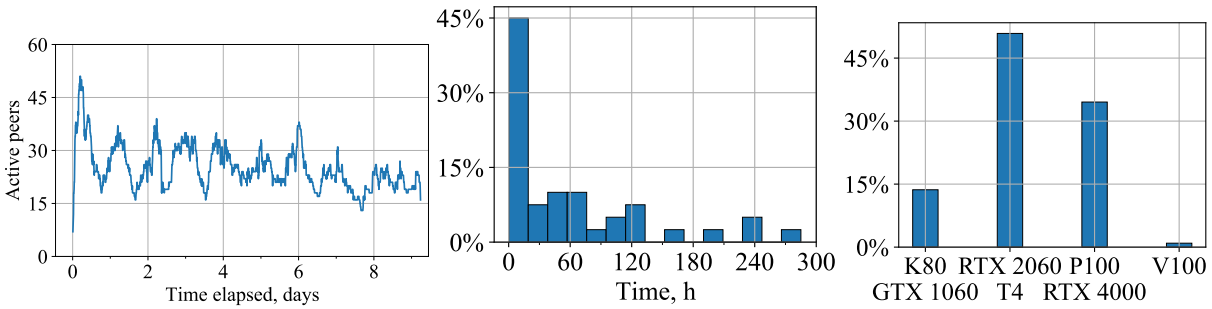


Figure 12: ALBERT training convergence in different setups.



(a) Worker count dynamics. (b) Participation time distribution. (c) Hardware statistics.

Figure 13: Collaborative experiment summary.

Finally, we conducted a real-world collaborative experiment by pretraining a version of ALBERT on the Bengali-language part of the OSCAR [33] dataset. We recruited 40 volunteers who joined the experiment from their own computers with GPUs and free cloud instances and named the resulting model sahajBERT in a collective vote. Overall, the volunteers participated in the experiment from 91 different nodes, on average 15–35 of which were online at the same time. The median contributed time was approximately 36 hours; additional volunteer statistics collected during the experiment are available in Figure 13. The model took approximately 8 days to converge, which was 1.8x faster than the single-node multi-GPU baseline we trained to verify equivalent convergence; see Figure 14 for an illustration.

We also compared the results of finetuning sahajBERT on two downstream datasets in Bengali with several existing baselines. The datasets we considered were WikiANN [15] and News Category Classification from IndicGLUE [26]; our baselines were monolingual bnRoBERTa [25], as well as multilingual IndicBERT [26] and XLM-RoBERTa [56]. The results of this comparison are given in Table 3: even though sahajBERT used unstable resources of volunteers instead of a dedicated supercomputer for pretraining, it achieved results that are competitive or even exceed those of highly capable baselines.

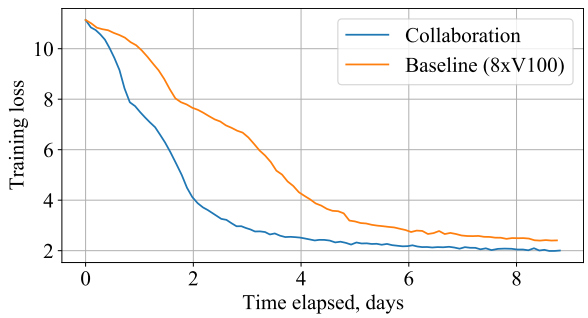


Figure 14: Training progress of sahajBERT.

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

Table 3: Downstream evaluation results.

4 Conclusion

In this section, we summarize the main contributions of the work. The results of this work are a set of methods for large-scale decentralized deep learning across unstable heterogeneous nodes with slow connectivity — in particular, across the computers of volunteers.

1. We proposed *Decentralized Mixture-of-Experts* (DMoE), a neural network layer designed for fault tolerance and significant numbers of peers with different memory capacities. In addition, we designed *Learning@home* — a system for deep learning in the setting of volunteer computing. It uses gradient checkpointing and distributed hash tables to prevent any single point of failure in the network and asynchronous request queuing to maximize the training throughput. We empirically validated that Learning@home maintains high training throughput both with emulated network latency and in realistic cross-region cloud training conditions. Finally, we trained neural networks with DMoE layers with node failures and high latency on two example machine learning tasks, showing that they converge to the same quality as regular dense models with a comparable parameter count.

2. We proposed *Moshpit All-Reduce*, a decentralized iterative method for averaging model gradients or parameters that uses efficient communication primitives as its foundation yet can be used in volatile networks. This method can be combined with standard stochastic optimization algorithms to obtain *Moshpit SGD* — a distributed training method suitable for training with unreliable devices. In experiments on synthetic data, Moshpit All-Reduce achieves the smallest inter-node distortion within a given number of iterations when compared to popular aggregation methods used in prior work. Moshpit SGD outperforms the baselines on two large-scale pretraining tasks by up to 1.5 times in terms of wall clock time until convergence. We also outlined the *dynamic load balancing* approach that solves a linear programming problem to assign varying workloads to peers with different network bandwidths.

3. We proposed *Distributed Deep Learning in Open Collaborations*, a practical method for collaborative deep learning. It leverages an *adaptive averaging strategy* to account for uneven communication and communication performance, as well as large-batch training with *global gradient accumulation* over the entire network to account for both dynamic peer participation and the presence of straggler nodes. We also ran a real-world collaborative pretraining experiment with a community of volunteers and obtained *sahajBERT* — a Bengali-language masked language model based on the ALBERT architecture. At the time of training, this model achieved similar downstream task performance to state-of-the-art models despite being trained on volunteer hardware instead of a GPU cluster.

References

1. A unified theory of decentralized SGD with changing topology and local updates / A. Koloskova [et al.] // International Conference on Machine Learning. — PMLR. 2020. — P. 5381–5393.
2. Adaptive Mixtures of Local Experts / R. A. Jacobs [et al.] // Neural Computation. — Cambridge, MA, USA, 1991. — Mar. — Vol. 3, no. 1. — P. 79–87. — ISSN 0899-7667. — DOI: 10.1162/neco.1991.3.1.79. — URL: https://doi.org/10.1162/neco.1991.3.1.79.
3. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations / Z.-Z. Lan [et al.] // ArXiv. — 2019. — Vol. abs/1909.11942.
4. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books / Y. Zhu [et al.] // Proceedings of the IEEE international conference on computer vision. — 2015. — P. 19–27.
5. Amazon EC2 Spot Instances. — URL: https://aws.amazon.com/ec2/spot/.
6. *Andersen E. D.*, *Andersen K. D.* The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm // Applied Optimization. — Springer US, 2000. — P. 197–232. — DOI: 10.1007/978-1-4757-3216-0_8. — URL: https://doi.org/10.1007%2F978-1-4757-3216-0_8.
7. *Anderson D. P.* Boinc: A system for public-resource computing and storage // Fifth IEEE/ACM international workshop on grid computing. — IEEE. 2004. — P. 4–10.
8. *Arjevani Y.*, *Shamir O.*, *Srebro N.* A tight convergence analysis for stochastic gradient descent with delayed updates // Algorithmic Learning Theory. — PMLR. 2020. — P. 111–132.
9. Asynchronous Decentralized Parallel Stochastic Gradient Descent / X. Lian [et al.] // Proceedings of the 35th International Conference on Machine Learning. Vol. 80 / ed. by J. Dy, A. Krause. — PMLR, 10–15 Jul/2018. — P. 3043–3052. — (Proceedings of Machine Learning Research). — URL: https://proceedings.mlr.press/v80/lian18a.html.
10. Attention is All you Need / A. Vaswani [et al.] // Advances in Neural Information Processing Systems 30 / ed. by I. Guyon [et al.]. — Curran Associates, Inc., 2017. — P. 5998–6008. — URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

11. Beyond hierarchies: Design considerations for distributed caching on the internet : tech. rep. / R. Tewari [et al.] ; Citeseer.

12. Breaking (Global) Barriers in Parallel Stochastic Optimization with Wait-Avoiding Group Averaging / S. Li [et al.] // IEEE Transactions on Parallel and Distributed Systems. — 2020. — P. 1–1. — ISSN 2161-9883. — DOI: 10.1109/tpds.2020.3040606. — URL: http://dx.doi.org/10.1109/TPDS.2020.3040606.

13. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent / X. Lian [et al.] // Advances in Neural Information Processing Systems. — 2017. — P. 5330–5340.

14. Communication efficient decentralized training with multiple local updates / X. Li [et al.] // arXiv preprint arXiv:1910.09126. — 2019. — Vol. 5.

15. Cross-lingual Name Tagging and Linking for 282 Languages / X. Pan [et al.] // Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. — 07/2017. — P. 1946–1958. — DOI: 10.18653/v1/P17-1178. — URL: https://www.aclweb.org/anthology/P17-1178.

16. Deep Residual Learning for Image Recognition / K. He [et al.] // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). — 2015. — P. 770–778.

17. Einstein@Home All-sky Search for Continuous Gravitational Waves in LIGO O2 Public Data / B. Steltner [et al.] // The Astrophysical Journal. — 2021. — Mar. — Vol. 909, no. 1. — P. 79. — ISSN 1538-4357. — DOI: 10.3847/1538-4357/abc7c9. — URL: http://dx.doi.org/10.3847/1538-4357/abc7c9.

18. Emergent abilities of large language models / J. Wei [et al.] // arXiv preprint arXiv:2206.07682. — 2022.

19. Folding@home gets 1.5+ Exaflops to Fight COVID-19. — Accessed: 2021-05-20. https://blogs.nvidia.com/blog/2020/04/01/foldingathome-exaflop-coronavirus/.

20. Folding@home update on SARS-CoV-2 (10 MAR 2020). — https://foldingathome.org/covid19/(accessed on June 4, 2020).

21. Google Cloud Spot VMs pricing. — URL: https://cloud.google.com/compute/docs/instances/spot#pricing.

22. Gpipe: Efficient training of giant neural networks using pipeline parallelism / Y. Huang [et al.] // Advances in Neural Information Processing Systems. — 2019. — P. 103–112.

23. Gradient-based learning applied to document recognition / Y. LeCun [et al.] // Proceedings of the IEEE. — 1998. — Vol. 86, no. 11. — P. 2278–2324.

24. ImageNet: A Large-Scale Hierarchical Image Database / J. Deng [et al.] // CVPR09. — 06/2009. — P. 248–255. — DOI: 10.1109/CVPR.2009.5206848.

25. Indic-Transformers: An Analysis of Transformer Language Models for Indian Languages / K. Jain [et al.] // arXiv preprint arXiv:2011.02323. — 2020.

26. IndicNLPSuite: Monolingual Corpora, Evaluation Benchmarks and Pre-trained Multilingual Language Models for Indian Languages / D. Kakwani [et al.] // Findings of the Association for Computational Linguistics: EMNLP 2020. — 11/2020. — P. 4948–4961. — DOI: 10.18653/v1/2020.findings-emnlp.445. — URL: https://www.aclweb.org/anthology/2020.findings-emnlp.445.

27. Language Models are Few-Shot Learners / T. B. Brown [et al.] // arXiv preprint arXiv:2005.14165. — 2020. — Vol. 33. — P. 1877–1901. — URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

28. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes / Y. You [et al.] // International Conference on Learning Representations. — 2020. — URL: https://openreview.net/forum?id=Syx4wnEtvH.

29. Large Memory Layers with Product Keys / G. Lample [et al.] // Advances in Neural Information Processing Systems 32 / ed. by H. Wallach [et al.]. — Curran Associates, Inc., 2019. — P. 8546–8557. — URL: http://papers.nips.cc/paper/9061-large-memory-layers-with-product-keys.pdf.

30. *Li M.* Scaling Distributed Machine Learning with the Parameter Server // Proceedings of the 2014 International Conference on Big Data Science and Computing. — Beijing, China : Association for Computing Machinery, 2014. — (BigDataScience '14). — ISBN 9781450328913. — DOI: 10.1145/2640087.2644155. — URL: https://doi.org/10.1145/2640087.2644155.

31. *Maymounkov P.*, *Mazieres D.* Kademlia: A peer-to-peer information system based on the xor metric // International Workshop on Peer-to-Peer Systems. — Springer. 2002. — P. 53–65.

32. Mixed Precision Training / P. Micikevicius [et al.] // International Conference on Learning Representations. — 2018. — URL: https://openreview.net/forum?id=r1gs9JgRZ.

33. *Ortiz Suárez P. J.*, *Romary L.*, *Sagot B.* A Monolingual Approach to Contextualized Word Embeddings for Mid-Resource Languages // Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. — 07/2020. — P. 1703–1714. — URL: https://www.aclweb.org/anthology/2020.acl-main.156.

34. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer / N. Shazeer [et al.] // arXiv preprint arXiv:1701.06538. — 2017.

35. PaLM: Scaling Language Modeling with Pathways / A. Chowdhery [et al.]. — 2022. — arXiv: 2204.02311 [cs.CL].

36. Parallelized Stochastic Gradient Descent / M. Zinkevich [et al.] // Advances in Neural Information Processing Systems. Vol. 23 / ed. by J. Lafferty [et al.]. — Curran Associates, Inc., 2010. — P. 2595–2603. — URL: https://proceedings.neurips.cc/paper/2010/file/abea47ba24142ed16b7d8fbf2c740e0d-Paper.pdf.

37. *Patarasuk P.*, *Yuan X.* Bandwidth Optimal All-Reduce Algorithms for Clusters of Workstations // J. Parallel Distrib. Comput. — USA, 2009. — Feb. — Vol. 69, no. 2. — P. 117–124. — ISSN 0743-7315. — DOI: 10.1016/j.jpdc.2008.09.002. — URL: https://doi.org/10.1016/j.jpdc.2008.09.002.

38. Pointer Sentinel Mixture Models / S. Merity [et al.] // arXiv preprint arXiv:1609.07843. — 2017.

39. *Popel M.*, *Bojar O.* Training Tips for the Transformer Model // The Prague Bulletin of Mathematical Linguistics. — 2018. — Mar. — Vol. 110. — DOI: 10.2478/pralin-2018-0002.

40. PyTorch Distributed: Experiences on Accelerating Data Parallel Training / S. Li [et al.] // Proc. VLDB Endow. — 2020. — Aug. — Vol. 13, no. 12. — P. 3005–3018. — ISSN 2150-8097. — DOI: 10.14778/3415478.3415530. — URL: https://doi.org/10.14778/3415478.3415530.

41. PyTorch: An imperative style, high-performance deep learning library / A. Paszke [et al.] // Advances in Neural Information Processing Systems. — 2019. — P. 8024–8035.

42. *Rabenseifner R.* Optimization of Collective Reduction Operations // Computational Science - ICCS 2004 / ed. by M. Bubak [et al.]. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2004. — P. 1–9. — ISBN 978-3-540-24685-5.

43. Randomized gossip algorithms / S. Boyd [et al.] // IEEE transactions on information theory. — 2006. — Vol. 52, no. 6. — P. 2508–2530.

44. *Sack P.*, *Gropp W.* Collective Algorithms for Multiported Torus Networks // ACM Trans. Parallel Comput. — New York, NY, USA, 2015. — Feb. — Vol. 1, no. 2. — ISSN 2329-4949. — DOI: 10.1145/2686882. — URL: https://doi.org/10.1145/2686882.

45. Scaling Laws for Autoregressive Generative Modeling / T. Henighan [et al.]. — 2020. — arXiv: 2010.14701 [cs.LG].

46. Scaling Laws for Neural Language Models / J. Kaplan [et al.]. — 2020. — arXiv: 2001.08361 [cs.LG].

47. *Sergeev A.*, *Balso M. D.* Horovod: fast and easy distributed deep learning in TensorFlow // arXiv preprint arXiv:1802.05799. — 2018.

48. SETI@home: An Experiment in Public-Resource Computing / D. Anderson [et al.] // Commun. ACM. — 2002. — Nov. — Vol. 45. — P. 56–61. — DOI: 10.1145/581571.581573.

49. SlowMo: Improving Communication-Efficient Distributed SGD with Slow Momentum / J. Wang [et al.] // International Conference on Learning Representations. — 2020. — URL: https://openreview.net/forum?id=SkxJ8REYPH.

50. *Stephen Merity et al. 2.* Wikitext-2. —. — URL: https://arxiv.org/abs/1609.07843.

51. *Stich S. U.* Local SGD Converges Fast and Communicates Little // International Conference on Learning Representations (ICLR). — 2019. — arXiv:1805.09767. — URL: https://arxiv.org/abs/1805.09767.

52. Stochastic Gradient Push for Distributed Deep Learning / M. Assran [et al.] // Proceedings of the 36th International Conference on Machine Learning. Vol. 97. — 06/2019. — P. 344–353. — (Proceedings of Machine Learning Research).

53. Stochastic Gradient Push for Distributed Deep Learning / M. Assran [et al.] // Proceedings of the 36th International Conference on Machine Learning. Vol. 97 / ed. by K. Chaudhuri, R. Salakhutdinov. — PMLR, 09–15 Jun/2019. — P. 344–353. — (Proceedings of Machine Learning Research). — URL: http://proceedings.mlr.press/v97/assran19a.html.

54. Training Compute-Optimal Large Language Models / J. Hoffmann [et al.]. — 2022. — arXiv: 2203.15556 [cs.CL].

55. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context / Z. Dai [et al.] // Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. — 2019. — P. 2978–2988.

56. Unsupervised Cross-lingual Representation Learning at Scale / A. Conneau [et al.] // Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. — 07/2020. — P. 8440–8451. — DOI: 10.18653/v1/2020.acl-main.747. — URL: https://www.aclweb.org/anthology/2020.acl-main.747.

57. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments / M. Caron [et al.] // Advances in Neural Information Processing Systems. Vol. 33. — 2020. — P. 9912–9924.

58. *You Y.*, *Gitman I.*, *Ginsburg B.* Large Batch Training of Convolutional Networks // arXiv preprint arXiv:1708.03888. — 2017.

59. ZeRO-Offload: Democratizing Billion-Scale Model Training / J. Ren [et al.] // arXiv preprint arXiv:2101.06840. — 2021.